UCRL-SM-212121

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

sundialsTB, a Matlab Interface to SUNDIALS

R. Serban

May 10, 2005

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

SUNDIALSTB, a MATLAB Interface to SUNDIALS

Radu Serban
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

May 2005

UCRL-SM-000000

Contents

1 Introduction

SUNDIALS [2], SUite of Nonlinear and DIfferential/ALgebraic equation Solvers, is a family of software tools for integration of ODE and DAE initial value problems and for the solution of nonlinear systems of equations. It consists of CVODE, IDA, and KINSOL, and variants of these with sensitivity analysis capabilities.

SUNDIALSTB is a collection of MATLAB functions which provide interfaces to the SUNDIALS solvers.

The core of each MATLAB interface in SUNDIALSTB is a single MEX file which interfaces to the various user-callable functions for that solver. However, this MEX file should not be called directly, but rather through the user-callable functions provided for each MATLAB interface.

A major design principle for SUNDIALSTB was to provide an interface that is, as much as possible, equally familiar to users of both the SUNDIALS codes and MATLAB. Moreover, we tried to keep the number of user-callable functions to a minimum. For example, the CVODES MATLAB interface contains only 9 such functions, 3 of which interface solely to the adjoint sensitivity module in CVODES. In tune with the MATLAB ODESET function, optional solver inputs in SUNDIALSTB are specified through a single function (`CvodeSetOptions` for CVODES). However, unlike the ODE solvers in MATLAB, we have kept the more flexible SUNDIALS model in which a separate "solve" function (`CVodeSolve` for CVODES) must be called to return the solution at a desired output time. Solver statistics, as well as optional outputs (such as solution and solution derivatives at additional times) can be obtained at any time with calls to separate functions (`CVodeGetStats` and `CVodeGet` for CVODES).

This document provides a complete documentation for the SUNDIALSTB functions. For additional details on the methods and underlying SUNDIALS software consult also the coresponding SUNDIALS user guides [3, 1].

1.1 Notes

The version numbers for the MATLAB interfaces correspond to those of the corresponding SUNDIALS solver with wich the interface is compatible.

1.2 Requirements

Each interface module in SUNDIALSTB requires the appropriate version of the corresponding SUNDIALS solver. For parallel support, SUNDIALSTB depends on MPITB with LAM v > 7.1.1 (for MPI-2 spawning feature).

1.3 Installation

1. Install the appropriate version of the SUNDIALS solver(s).
2. Modify `Makefile` (`SUNDIALS` location) in the `mex` directory and compile
3. Optionally, for parallel support, install and configure LAM (local copy, since typical installations only install static libraries) and MPITB
4. Add the following paths to your MATLAB `startup.m` script:
    - `sundialsTB/cvodes` and `sundialsTB/mex/cvm` for CVODES
    - `sundialsTB/kinsol` and `sundialsTB/mex/kim` for KINSOL
    - `sundialsTB/nvector` and `sundialsTB/mex/nvm` for NVECTOR operations
    - `sundialsTB/putils` for `mpirun` function
5. In MATLAB, try:
    - `help cvodes`
    - `help kinsol`

- help nvector
- help putils

1.4 Links

The required software packages can be obtained from the following addresses.

SUNDIALS	http://www.llnl.gov/CASC/sundials
MPITB	http://atc.ugr.es/javier-bin/mpitb_eng
LAM	http://www.lam-mpi.org/

2 MATLAB Interface to CVODES

The MATLAB interface to CVODES provides access to all functionality of the CVODES solver, including IVP simulation and sensitvity analysis (both forward and adjoint).

The interface consists of 9 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions and the types of user-supplied functions are listed in Table 1 and fully documented later in this section. For more in depth details, consult also the CVODES user guide [3].

To illustrate the use of the CVODES MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with CVODES.

Table 1: CVODES MATLAB interface functions

Functions	CVodeSetOptions	creates an options structure for CVODES.
	CVodeMalloc	allocates and initializes memory for CVODES.
	CVodeMallocB	allocates and initializes backward memory for CVODES.
	CVode	integrates the ODE.
	CVodeB	integrates the backward ODE.
	CVodeGetStats	returns statistics for the CVODES solver.
	CVodeGetStatsB	returns statistics for the backward CVODES solver.
	CVodeGet	extracts data from CVODES memory.
	CVodeFree	deallocates memory for the CVODES solver.
	CVodeMonitor	sample monitoring function.
Function types	CVRhsFn	RHS function
	CVRootFn	root-finding function
	CVQuadRhsFn	quadrature RHS function
	CVDenseJacFn	dense Jacobian function
	CVBandJacFn	banded Jacobian function
	CVJactimesVecFn	Jacobian times vector function
	CVPrecSetupFn	preconditioner setup function
	CVPrecSolveFn	preconditioner solve function
	CVGlocalFn	RHS approximation function (BBDPre)
	CVGcommFn	communication function (BBDPre)
	CVSensRhsFn	sensitivity RHS function
	CVSensRhs1Fn	sensitivity RHS function (single)
	CVMonitorFn	monitoring function

2.1 Interface functions

CVodeSetOptions

Purpose

CVodeSetOptions creates an options structure for CVODES.

Synopsis

```
function options = CVodeSetOptions(varargin)
```

Description

CVodeSetOptions creates an options structure for CVODES.

```
   Usage: OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
          OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)
          OPTIONS = CVodeSetOptions(OLDOPTIONS,NEWOPTIONS)

   OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates
   a CVODES options structure OPTIONS in which the named properties have
   the specified values. Any unspecified properties have default values.
   It is sufficient to type only the leading characters that uniquely
   identify the property. Case is ignored for property names.

   OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an
   existing options structure OLDOPTIONS.

   OPTIONS = CVodeSetOptions(OLDOPTIONS,NEWOPTIONS) combines an existing
   options structure OLDOPTIONS with a new options structure NEWOPTIONS.
   Any new properties overwrite corresponding old properties.

   CVodeSetOptions with no input arguments displays all property names
   and their possible values.

CVodeSetOptions properties
(See also the CVODES User Guide)

Adams - Use Adams linear multistep method [ on | off ]
   This property specifies whether the Adams method is to be used instead
   of the default Backward Differentiation Formulas (BDF) method.
   The Adams method is recommended for non-stiff problems, while BDF is
   recommended for stiff problems.
NonlinearSolver - Type of nonlinear solver used [ Functional | Newton ]
   The 'Functional' nonlinear solver is best suited for non-stiff
   problems, in conjunction with the 'Adams' linear multistep method,
   while 'Newton' is better suited for stiff problems, using the 'BDF'
   method.
RelTol - Relative tolerance [ positive scalar | 1e-4 ]
   RelTol defaults to 1e-4 and is applied to all components of the solution
   vector. See AbsTol.
AbsTol - Absolute tolerance [ positive scalar or vector | 1e-6 ]
```

```
   The relative and absolute tolerances define a vector of error weights
   with components
     ewt(i) = 1/(RelTol*|y(i)| + AbsTol)    if AbsTol is a scalar
     ewt(i) = 1/(RelTol*|y(i)| + AbsTol(i)) if AbsTol is a vector
   This vector is used in all error and convergence tests, which
   use a weighted RMS norm on all error-like vectors v:
     WRMSnorm(v) = sqrt( (1/N) sum(i=1..N) (v(i)*ewt(i))^2 ),
   where N is the problem dimension.
MaxNumSteps - Maximum number of steps [positive integer | 500]
   CVode will return with an error after taking MaxNumSteps internal steps
   in its attempt to reach the next output time.
InitialStep - Suggested initial stepsize [ positive scalar ]
   By default, CVode estimates an initial stepsize h0 at the initial time
   t0 as the solution of
     WRMSnorm(h0^2 ydd / 2) = 1
   where ydd is an estimated second derivative of y(t0).
MaxStep - Maximum stepsize [ positive scalar | inf ]
   Defines an upper bound on the integration step size.
MinStep - Minimum stepsize [ positive scalar | 0.0 ]
   Defines a lower bound on the integration step size.
MaxOrder - Maximum method order [ 1-12 for Adams, 1-5 for BDF | 5 ]
   Defines an upper bound on the linear multistep method order.
StopTime - Stopping time [ scalar ]
   Defines a value for the independent variable past which the solution
   is not to proceed.
RootsFn - Rootfinding function [ function ]
   To detect events (roots of functions), set this property to the event
   function. See CVRootFn.
NumRoots - Number of root functions [ integer | 0 ]
   Set NumRoots to the number of functions for which roots are monitored.
   If NumRoots is 0, rootfinding is disabled.
StabilityLimDet - Stability limit detection algorithm [ on | off ]
   Flag used to turn on or off the stability limit detection algorithm
   within CVODES. This property can be used only with the BDF method.
   In this case, if the order is 3 or greater and if the stability limit
   is detected, the method order is reduced.

LinearSolver - Linear solver type [ Diag | Band | GMRES | BiCGStab | Dense ]
   Specifies the type of linear solver to be used for the Newton nonlinear
   solver (see NonlinearSolver). Valid choices are: Dense (direct, dense
   Jacobian), Band (direct, banded Jacobian), Diag (direct, diagonal Jacobian),
   GMRES (iterative, scaled preconditioned GMRES), BiCGStab (iterative, scaled
   preconditioned stabilized BiCG). The GMRES and BiCGStab are matrix-free
   linear solvers.
JacobianFn - Jacobian function [ function ]
   This propeerty is overloaded. Set this value to a function that returns
   Jacobian information consistent with the linear solver used (see Linsolver).
   If not specified, CVODES uses difference quotient approximations.
   For the Dense linear solver, JacobianFn must be of type CVDenseJacFn and
   must return a dense Jacobian matrix. For the Band linear solver, JacobianFn
   must be of type CVBandJacFn and must return a banded Jacobian matrix.
   For the iterative linear solvers, GMRES and BiCGStab, JacobianFn must be
   of type CVJacTimesVecFn and must return a Jacobian-vector product. This
   property is not used for the Diag linear solver.
```

PrecType - Preconditioner type [Left | Right | Both | None]
Specifies the type of user preconditioning to be done if an iterative linear solver, GMRES or BiCGStab, is used (see LinSolver). PrecType must be one of the following: 'None', 'Left', 'Right', or 'Both', corresponding to no preconditioning, left preconditioning only, right preconditioning only, and both left and right preconditioning, respectively.

PrecModule - Preconditioner module [BandPre | BBDPre | UserDefined]
If the PrecModule = 'UserDefined', then the user must provide at least a preconditioner solve function (see PrecSolveFn)
CVODES provides the following two general-purpose preconditioner modules:
BandPre provide a band matrix preconditioner based on difference quotients of the ODE right-hand side function. The user must specify the lower and upper half-bandwidths through the properties LowerBwidth and UpperBwidth, respectively.
BBDPre can be only used with parallel vectors. It provide a preconditioner matrix that is block-diagonal with banded blocks. The blocking corresponds to the distribution of the dependent variable vector y among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor) of a given function g(t,y) approximating f(t,y) (see GlocalFn). The blocks are generated by a difference quotient scheme on each processor independently. This scheme utilizes an assumed banded structure with given half-bandwidths, mldq and mudq (specified through LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian block kept by the scheme has half-bandwiths ml and mu (specified through LowerBwidth and UpperBwidth), which may be smaller.

PrecSetupFn - Preconditioner setup function [function]
If PrecType is not 'None', PrecSetupFn specifies an optional function which, together with PrecSolve, defines left and right preconditioner matrices (either of which can be trivial), such that the product P1*P2 is an aproximation to the Newton matrix. PrecSetupFn must be of type CVPrecSetupFn.

PrecSolveFn - Preconditioner solve function [function]
If PrecType is not 'None', PrecSolveFn specifies a required function which must solve a linear system Pz = r, for given r. PrecSolveFn must be of type CVPrecSolveFn.

KrylovMaxDim - Maximum number of Krylov subspace vectors [integer | 5]
Specifies the maximum number of vectors in the Krylov subspace. This property is used only if an iterative linear solver, GMRES or BiCGStab, is used (see LinSolver).

GramSchmidtType - Gram-Schmidt orthogonalization [Classical | Modified]
Specifies the type of Gram-Schmidt orthogonalization (classical or modified). This property is used only if the GMRES linear solver is used (see LinSolver).

GlocalFn - Local right-hand side approximation funciton for BBDPre [function]
If PrecModule is BBDPre, GlocalFn specifies a required function that evaluates a local approximation to the ODE right-hand side. GlocalFn must be of type CVGlocFn.

GcommFn - Inter-process communication function for BBDPre [function]
If PrecModule is BBDPre, GcommFn specifies an optional function to perform any inter-process communication required for the evaluation of GlocalFn. GcommFn must be of type CVGcommFn.

LowerBwidth - Jacobian/preconditioner lower bandwidth [integer | 0]
This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the lower half-bandwidth of the band Jacobian approximation. If one of the two iterative linear solvers, GMRES or BiCGStab, is used (see LinSolver) and if the BBDPre preconditioner module in CVODES is used

(see PrecModule), it specifies the lower half-bandwidth of the retained banded approximation of the local Jacobian block. If the BandPre preconditioner module (see PrecModule) is used, it specifies the lower half-bandwidth of the band preconditioner matrix. LowerBwidth defaults to 0 (no sub-diagonals).

UpperBwidth - Jacobian/preconditioner upper bandwidth [integer | 0]
This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the upper half-bandwidth of the band Jacobian approximation. If one of the two iterative linear solvers, GMRES or BiCGStab, is used (see LinSolver) and if the BBDPre preconditioner module in CVODES is used (see PrecModule), it specifies the upper half-bandwidth of the retained banded approximation of the local Jacobian block. If the BandPre preconditioner module (see PrecModule) is used, it specifies the upper half-bandwidth of the band preconditioner matrix. UpperBwidth defaults to 0 (no super-diagonals).

LowerBwidthDQ - BBDPre preconditioner DQ lower bandwidth [integer | 0]
Specifies the lower half-bandwidth used in the difference-quotient Jacobian approximation for the BBDPre preconditioner (see PrecModule).

UpperBwidthDQ - BBDPre preconditioner DQ upper bandwidth [integer | 0]
Specifies the upper half-bandwidth used in the difference-quotient Jacobian approximation for the BBDPre preconditioner (see PrecModule).

Quadratures - Quadrature integration [on | off]
Enables or disables quadrature integration.

QuadRhsFn - Quadrature right-hand side function [function]
Specifies the user-supplied function to evaluate the integrand for quadrature computations. See CVQuadRhsfn.

QuadInitCond - Initial conditions for quadrature variables [vector]
Specifies the initial conditions for quadrature variables.

QuadErrControl - Error control strategy for quadrature variables [on | off]
Specifies whether quadrature variables are included in the error test.

QuadRelTol - Relative tolerance for quadrature variables [scalar 1e-4]
Specifies the relative tolerance for quadrature variables. This parameter is used only if QuadErrCon=on.

QuadAbsTol - Absolute tolerance for quadrature variables [scalar or vector 1e-6]
Specifies the absolute tolerance for quadrature variables. This parameter is used only if QuadErrCon=on.

SensAnalysis - Sensitivity anlaysis [FSA | ASA | off]
Enables sensitivity analysis computations. CVODES can perform both Forward Sensitivity Analysis (FSA) and Adjoint Sensitivity Analysis (ASA).

FSAInitCond - Initial conditions for sensitivity variables [matrix]
Specifies the initial conditions for sensitivity variables. FSAInitcond must be a matrix with N rows and Ns columns, where N is the problem dimension and Ns the number of sensitivity systems.

FSAMethod - FSA solution method [Simultaneous | Staggered1 | Staggered]
Specifies the FSA method for treating the nonlinear system solution for sensitivity variables. In the simultaneous case, the nonlinear systems for states and all sensitivities are solved simultaneously. In the Staggered case, the nonlinear system for states is solved first and then the nonlinear systems for all sensitivities are solved at the same time. Finally, in the Staggered1 approach all nonlinear systems are solved in a sequence (in this case, the sensitivity right-hand sides must be available for each sensitivity system sepaately - see SensRHS and SensRHStype).

FSAParamField - Problem parameters [string]
Specifies the name of the field in the user data structure (passed as an argument to CVodeMalloc) in which the nominal values of the problem parameters are stored. This property is used only if CVODES will use difference quotient approximations to the sensitivity right-hand sides (see SensRHS and SensRHStype).

FSAParamList - Parameters with respect to which FSA is performed [integer vector]
Specifies a list of Ns parameters with respect to which sensitivities are to be computed. This property is used only if CVODES will use difference-quotient approximations to the sensitivity right-hand sides (see SensRHS and SensRHStype). Its length must be Ns, consistent with the number of columns of FSAinitCond.

FSAParamScales - Order of magnitude for problem parameters [vector]
Provides order of magnitude information for the parameters with respect to which sensitivities are computed. This information is used if CVODES approximates the sensitivity right-hand sides (see SensRHS) or if CVODES estimates integration tolerances for the sensitivity variables (see FSAReltol and FSAAbsTol).

FSARelTol - Relative tolerance for sensitivity variables [positive scalar]
Specifies the scalar relative tolerance for the sensitivity variables. See FSAAbsTol.

FSAAbsTol - Absolute tolerance for sensitivity variables [row-vector or matrix]
Specifies the absolute tolerance for sensitivity variables. FSAAbsTol must be either a row vector of dimension Ns, in which case each of its components is used as a scalar absolute tolerance for the coresponding sensitivity vector, or a N x Ns matrix, in which case each of its columns is used as a vector of absolute tolerances for the corresponding sensitivity vector.
By default, CVODES estimates the integration tolerances for sensitivity variables, based on those for the states and on the order of magnitude information for the problem parameters specified through ParamScales.

FSAErrControl - Error control strategy for sensitivity variables [on | off]
Specifies whether sensitivity variables are included in the error control test. Note that sensitivity variables are always included in the nonlinear system convergence test.

FSARhsFn - Sensitivity right-hand side function [function]
Specifies a user-supplied function to evaluate the sensitivity right-hand sides. This property is overloaded. The type of this function must be either CVSensRhsFn (if it returns the righ-hand sides for all sensitivity systems at once) or CVSensRhs1Fn (if it returns the right-hand side for the i-th sensitivity). See SensRHStype. By default, CVODES uses an internal difference-quotient function to approximate the sensitivity right-hand sides.

FSARhsType - Type of the sensitivity right-hand side function [All | One]
Specifies the type of the function which computes the sensitivity right-hand sides. FSARhsType = 'All' indicates that FSARhsFn is of type CVSensRhsFn. FSARhsType = 'One' indicates that FSARhsFn is of type CVSensRhs1Fn. Note that either function type can be used with FSAMethod = 'Simultaneous' or with FSAMethod = 'Staggered', but only FSARhsType = 'One' is acceptable for FSAMethod = 'Staggered1'.

FSADQparam - Parameter for the DQ approx. of the sensi. RHS [scalar | 0.0]
Specifies the value which controls the selection of the difference-quotient scheme used in evaluating the sensitivity right-hand sides. This property is used only if CVODES will use difference-quotient approximations. The default value 0.0 indicates the use of the second-order centered directional derivative formula exclusively. Otherwise, the magnitude of FSADQparam and its sign (positive or negative) indicates whether this switching is done with regard

```
   to (centered or forward) finite differences, respectively.

ASANumDataPoints - Number of data points for ASA [ integer | 100 ]
   Specifies the (maximum) number of integration steps between two consecutive
   check points.
ASAInterpType - Type of interpolation [ Hermite ]
   Specifies the type of interpolation used for estimating the forward solution
   during the backward integration phase. At this time, the only option is
   'Hermite', specifying cubic Hermite interpolation.

MonitorFn - User-provied monitoring function [ function ]
   Specifies a function that is called after each successful integration step.
   This function must have type CVMonitorFn. A simple monitoring function,
   CVodeMonitor is provided with CVODES.
MonitorData - User-provied data for the monitoring function [ struct ]
   Specifies a data structure that is passed to the Monitor function every time
   it is called.

   See also
        CVRootFn, CVQuadRhsFn
        CVSensRhsFn, CVSensRhs1Fn
        CVDenseJacFn, CVBandJacFn, CVJacTimesVecFn
        CVPrecSetupFn, CVPrecSolveFn
        CVGlocalFn, CVGcommFn
        CVMonitorFn
```

CVodeMalloc

Purpose

CVodeMalloc allocates and initializes memory for CVODES.

Synopsis

```
function [] = CVodeMalloc(fct,t0,y0,varargin)
```

Description

CVodeMalloc allocates and initializes memory for CVODES.

```
   Usage: CVodeMalloc ( ODEFUN, T0, Y0 [, OPTIONS [, DATA] ] )

   ODEFUN   is a function defining the ODE right-hand side: y' = f(t,y).
            This function must return a vector containing the current
            value of the righ-hand side.
   T0       is the initial value of t.
   Y0       is the initial condition vector y(t0).
   OPTIONS  is an (optional) set of integration options, created with
            the CVodeSetOptions function.
   DATA     is (optional) problem data passed unmodified to all
            user-provided functions when they are called. For example,
            YD = ODEFUN(T,Y,DATA).

   See also: CVRhsFn
```

CVodeMallocB

Purpose

CVodeMallocB allocates and initializes backward memory for CVODES.

Synopsis

```
function [] = CVodeMallocB(fctB,tB0,yB0,varargin)
```

Description

CVodeMallocB allocates and initializes backward memory for CVODES.

```
Usage:   CVodeMallocB ( FCTB, TB0, YB0 [, OPTIONSB] )

FCTB     is a function defining the adjoint ODE right-hand side.
         This function must return a vector containing the current
         value of the adjoint ODE righ-hand side.
TB0      is the final value of t.
YB0      is the final condition vector yB(tB0).
OPTIONSB is an (optional) set of integration options, created with
         the CVodeSetOptions function.

See also: CVRhsFn
```

CVode

Purpose

CVode integrates the ODE.

Synopsis

```
function [status,t,y,varargout] = CVode(tout,itask)
```

Description

CVode integrates the ODE.

```
Usage: [STATUS, T, Y] = CVode ( TOUT, ITASK )
       [STATUS, T, Y, YS] = CVode ( TOUT, ITASK )
       [STATUS, T, Y, YQ] = CVode  (TOUT, ITASK )
       [STATUS, T, Y, YQ, YS] = CVode ( TOUT, ITASK )

If ITASK is 'Normal', then the solver integrates from its current internal
T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
Y(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
and returns in Y the solution at the new internal time. In this case, TOUT
is used only during the first call to CVode to determine the direction of
integration and the rough scale of the problem. In either case, the time
reached by the solver is returned in T. The 'NormalTstop' and 'OneStepTstop'
modes are similar to 'Normal' and 'OneStep', respectively, except that the
integration never proceeds past the value tstop.
```

```
If quadratures were computed (see CVodeSetOptions), CVode will return their
values at T in the vector YQ.

If sensitivity calculations were enabled (see CVodeSetOptions), CVode will
return their values at T in the matrix YS.

On return, STATUS is one of the following:
  0: CVode succeeded and no roots were found.
  1: CVode succeded and returned at tstop.
  2: CVode succeeded, and found one or more roots.
 -1: Illegal attempt to call before CVodeMalloc
 -2: One of the inputs to CVode is illegal. This includes the situation
     when a component of the error weight vectors becomes &lt; 0 during internal
     time-stepping.
 -4: The solver took mxstep internal steps but could not reach TOUT. The
     default value for mxstep is 500.
 -5: The solver could not satisfy the accuracy demanded by the user for some
     internal step.
 -6: Error test failures occurred too many times (MXNEF = 7) during one internal
     time step
     or occurred with |h| = hmin.
 -7: Convergence test failures occurred too many times (MXNCF = 10) during one
     internal time step or occurred with |h| = hmin.
 -9: The linear solver's setup routine failed in an unrecoverable manner.
-10: The linear solver's solve routine failed in an unrecoverable manner.


See also CVodeSetOptions, CVodeGetstats
```

CVodeB

Purpose

CVodeB integrates the backward ODE.

Synopsis

```
function [status,t,yB,varargout] = CVodeB(tout,itask)
```

Description

CVodeB integrates the backward ODE.

```
Usage:  [STATUS, T, YB] = CVodeB ( TOUT, ITASK )
        [STATUS, T, YB, YQB] = CVodeB ( TOUT, ITASK )

If ITASK is 'Normal', then the solver integrates from its current internal
T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns
YB(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step
and returns in YB the solution at the new internal time. In this case, TOUT
is used only during the first call to CVodeB to determine the direction of
integration and the rough scale of the problem. In either case, the time
reached by the solver is returned in T.
```

```
   If quadratures were computed (see CVodeSet), CVodeB will return their
   values at T in the vector YQB.

   On return, STATUS is one of the following:
     0: CVodeB succeeded and no roots were found.
    -2: One of the inputs to CVodeB is illegal.
    -4: The solver took mxstep internal steps but could not reach TOUT.
        The default value for mxstep is 500.
   -5:  The solver could not satisfy the accuracy demanded by the user for
        some internal step.
   -6:  Error test failures occurred too many times (MXNEF = 7) during one
        internal time step or occurred with |h| = hmin.
   -7:  Convergence test failures occurred too many times (MXNCF = 10) during
        one internal time step or occurred with |h| = hmin.
   -9:  The linear solver's setup routine failed in an unrecoverable manner.
  -10:  The linear solver's solve routine failed in an unrecoverable manner.
  -101: Illegal attempt to call before initializing adjoint sensitivity
        (see CVodeMalloc).
  -104: Illegal attempt to call before CVodeMallocB.
  -108: Wrong value for TOUT.

   See also CVodeSetOptions, CVodeGetstatsB
```

CVodeGetStats

Purpose

CVodeGetStats returns run statistics for the CVODES solver.

Synopsis

```
function si = CVodeGetStats()
```

Description

```
CVodeGetStats returns run statistics for the CVODES solver.

   Usage: STATS = CVodeGetStats

Fields in the structure STATS

o nst      - number of integration steps
o nfe      - number of right-hand side function evaluations
o nsetups  - number of linear solver setup calls
o netf     - number of error test failures
o nni      - number of nonlinear solver iterations
o ncfn     - number of convergence test failures
o qlast    - last method order used
o qcur     - current method order
o h0used   - actual initial step size used
o hlast    - last step size used
o hcur     - current step size
o tcur     - current time reached by the integrator
```

```
o RootInfo - strucutre with rootfinding information
o QuadInfo - structure with quadrature integration statistics
o LSInfo   - structure with linear solver statistics
o FSAInfo  - structure with forward sensitivity solver statistics

If rootfinding was requested, the structure RootInfo has the following fields

o nge   - number of calls to the rootfinding function
o roots - array of integers (a value of 1 in the i-th component means that the
          i-th rootfinding function has a root (upon a return with status=2 from
          CVode).

If quadratures were present, the structure QuadInfo has the following fields

o nfQe  - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

The structure LSinfo has different fields, depending on the linear solver used.

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'Diag' linear solver

o name  - 'Diag'
o nfeDI - number of right-hand side function evaluations for difference-quotient
          Jacobian approximation

  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nfeB - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

o name  - 'GMRES' or 'BiCGStab'
o nli   - number of linear solver iterations
o npe   - number of preconditioner setups
o nps   - number of preconditioner solve function calls
o ncfl  - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nfeSG - number of right-hand side function evaluations for difference-quotient
          Jacobian-vector product approximation

If forward sensitivities were computed, the structure FSAInfo has the
following fields

o nfSe      - number of sensitivity right-hand side evaluations
```

```
o nfeS      - number of right-hand side evaluations for difference-quotient
              sensitivity right-hand side approximation
o nsetupsS  - number of linear solver setups triggered by sensitivity variables
o netfS     - number of error test failures for sensitivity variables
o nniS      - number of nonlinear solver iterations for sensitivity variables
o ncfnS     - number of convergence test failures due to sensitivity variables
o nniSTGR1  - number of nonlinear solver iterations for each sensitivity system
o ncfnSTGR1 - number of convergence test failures for each sensitivity system
```

CVodeGetStatsB

Purpose

CVodeGetStatsB returns run statistics for the backward CVODES solver.

Synopsis

```
function si = CVodeGetStatsB()
```

Description

```
CVodeGetStatsB returns run statistics for the backward CVODES solver.

   Usage: STATS = CVodeGetStatsB

Fields in the structure STATS

o nst - number of integration steps
o nfe - number of right-hand side function evaluations
o nsetups - number of linear solver setup calls
o netf - number of error test failures
o nni - number of nonlinear solver iterations
o ncfn - number of convergence test failures
o qlast - last method order used
o qcur - current method order
o h0used - actual initial step size used
o hlast - last step size used
o hcur - current step size
o tcur - current time reached by the integrator
o QuadInfo - structure with quadrature integration statistics
o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

If quadratures were present, the structure QuadInfo has the following fields

o nfQe - number of quadrature integrand function evaluations
o netfQ - number of error test failures for quadrature variables

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
```

```
        Jacobian approximation

  Fields in LSinfo for the 'Diag' linear solver

o name - 'Diag'
o nfeDI - number of right-hand side function evaluations for difference-quotient
          Jacobian approximation

  Fields in LSinfo for the 'Band' linear solver

o name - 'Band'
o njeB - number of Jacobian evaluations
o nfeB - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

o name - 'GMRES' or 'BiCGStab'
o nli - number of linear solver iterations
o npe - number of preconditioner setups
o nps - number of preconditioner solve function calls
o ncfl - number of linear system convergence test failures
o njeSG - number of Jacobian-vector product evaluations
o nfeSG -  number of right-hand side function evaluations for difference-quotient
          Jacobian-vector product approximation
```

CVodeGet

Purpose

CVodeGet extracts data from the CVODES solver memory.

Synopsis

```
function varargout = CVodeGet(key, varargin)
```

Description

```
CVodeGet extracts data from the CVODES solver memory.

   Usage: RET = CVodeGet ( KEY [, P1 [, P2] ... ])

   CVodeGet returns internal CVODES information based on KEY. For some values
   of KEY, additional arguments may be required and/or more than one output is
   returned.

   KEY is a string and should be one of:
    o DerivSolution - Returns a vector containing the K-th order derivative
       of the solution at time T. The time T and order K must be passed through
       the input arguments P1 and P2, respectively:
       DKY = CVodeGet('DerivSolution', T, K)
    o ErrorWeights - Returns a vector containing the error weights.
       EWT = CVodeGet('ErrorWeights')
    o CheckPointsInfo - Returns an array of structures with check point information.
```

```
    CK = CVodeGet('CheckPointInfo)
  o CurrentCheckPoint - Returns the address of the active check point
    ADDR = CVodeGet('CurrentCheckPoint');
  o DataPointInfo - Returns information stored for interpolation at the I-th data
    point in between the current check points. The index I must be passed through
    the agument P1.
    If the interpolation type was Hermite (see CVodeSetOptions), it returns two
    vectors, Y and YD:
    [Y, YD] = CVodeGet('DataPointInfo', I)
```

CVodeFree

Purpose

CVodeFree deallocates memory for the CVODES solver.

Synopsis

function [] = CVodeFree()

Description

CVodeFree deallocates memory for the CVODES solver.

```
   Usage:  CVodeFree
```

CVodeMonitor

Purpose

CVodeMonitor is a simple monitoring function example.

Synopsis

function [] = CVodeMonitor(call, time, sol, varargin)

Description

```
CVodeMonitor is a simple monitoring function example.
   To use it, set the Monitor property in CVodeSetOptions to
   'CVodeMonitor' or to @CVodeMonitor.

   With default settings, this function plots the evolution of the step
   size, method order, and various counters.

   Various properties can be changed from their default values by passing
   to CvodeSetOptions, through the property 'MonitorData', a structure
   MONDATA with any of the following fields. If a field is not defined,
   the corresponding default value is used.

   Fields in MONDATA structure:
     o stats [ true | false ]
         If true, CVodeMonitor reports the evolution of the step size and
         method order.
```

```
    o cntr [ true | false ]
        If true, CVodeMonitor reports the evolution of the following counters:
        nst, nfe, nni, netf, ncfn (see CVodeGetStats)
    o sol  [ true | false ]
        If true, CvodeMonitor plots all solution components (graphical mode only).
    o grph [ true | false ]
        If true, CvodeMonitor plots the evolutions of the above quantities.
        Otherwise, it prints to the screen.
    o updt [ integer | 50 ]
        CvodeMonitor update frequency.
    o select [ array of integers ]
        To plot only particular solution components, specify their indeces in
        the field select. If defined, it automatically sets sol=true. If not defined,
        but sol=true, CVodeMonitor plots all components (graphical mode only).
    o xaxis [ linear | log ]
        Type of the time axis for the stepsize, order, and counter plots
        (graphical mode only).
    o dir [ 1 | -1 ]
        Specifies forward or backward integration.

  See also CVodeSetOptions, CVMonitorFn
```

Source Code

```
function [] = CVodeMonitor(call, time, sol, varargin)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

persistent data
persistent first
persistent hf1 hf2 npl
persistent i
persistent t y h q nst nfe nni netf ncfn

if call == 0

  if nargin > 3
    data = varargin{1};
  end

  data = initialize_data(data, length(sol));

  first = true;
  if data.grph
    npl = 0;
    if data.stats
      npl = npl + 2;
    end
    if data.cntr
      npl = npl + 1;
    end
    if npl ~= 0
      hf1 = figure;
```

```
      end
    end
    if data.sol
      hf2 = figure;
      colormap(data.map);
    end

    i = 1;
    t = zeros(1,data.updt);
    if data.stats
      h = zeros(1,data.updt);
      q = zeros(1,data.updt);
    end
    if data.cntr
      nst = zeros(1,data.updt);
      nfe = zeros(1,data.updt);
      nni = zeros(1,data.updt);
      netf = zeros(1,data.updt);
      ncfn = zeros(1,data.updt);
    end
    if data.sol
      N = length(data.select);
      y = zeros(N,data.updt);
    end

    return;

end

% Load current statistics

if data.dir == 1
  si = CVodeGetStats;
else
  si = CVodeGetStatsB;
end

t(i) = si.tcur;

if data.stats
  h(i) = si.hlast;
  q(i) = si.qlast;
end

if data.cntr
  nst(i) = si.nst;
  nfe(i) = si.nfe;
  nni(i) = si.nni;
  netf(i) = si.netf;
  ncfn(i) = si.ncfn;
end

if data.sol
  N = length(data.select);
```

```
  for j = 1:N
    y(j,i) = sol(data.select(j));
  end
else
  N = 0;
end

% Finalize post

if call == 2
  if data.grph
    graphical_final(i,...
                    hf1, npl, data.stats, data.cntr, data.sol, data.dir,...
                    t, h, q, nst, nfe, nni, netf, ncfn,...
                    hf2, y, N, data.select);
  else
    text_final(i,data.stats,data.cntr,t,h,q,nst,nfe,nni,netf,ncfn);
  end
  return
end

% Is it time to post?

if i == data.updt

  if first
    if data.grph
      graphical_init(hf1, npl, data.stats, data.cntr, data.sol, data.dir,...
                     t, h, q, nst, nfe, nni, netf, ncfn,...
                     hf2, y, N, data.xaxis);
    else
      text_update(data.stats,data.cntr,t,h,q,nst,nfe,nni,netf,ncfn);
    end
    first = false;
  else
    if data.grph
      graphical_update(hf1, npl, data.stats, data.cntr, data.sol, data.dir,...
                       t, h, q, nst, nfe, nni, netf, ncfn,...
                       hf2, y, N);
    else
      text_update(data.stats,data.cntr,t,h,q,nst,nfe,nni,netf,ncfn);
    end
  end
  i = 1;
else
  i = i + 1;
end



%---------------------------------------------------------------------------

function data = initialize_data(data, N)

```

```
if ~isfield(data,'grph')
  data.grph = true;
end
if ~isfield(data,'updt')
  data.updt = 50;
end
if ~isfield(data,'stats')
  data.stats = true;
end
if ~isfield(data,'cntr')
  data.cntr = true;
end
if ~isfield(data,'sol')
  data.sol = false;
end
if ~isfield(data,'map')
  data.map = 'default';
end
if ~isfield(data,'select')
  data.select = [1:N];
else
  data.sol = true;
end
if ~isfield(data,'xaxis')
  data.xaxis = 'log';
end
if ~isfield(data,'dir')
  data.dir = 1;
end

if ~data.grph
  data.sol = false;
end

%----------------------------------------------------------------------------------

function [] = graphical_init(hf1, npl, stats, cntr, sol, dir,...
                             t, h, q, nst, nfe, nni, netf, ncfn,...
                             hf2, y, N, xaxis)

if npl ~= 0
  figure(hf1);
  pl = 0;
end

% Step size and order
if stats
  pl = pl+1;
  subplot(npl,1,pl)
  semilogy(t,abs(h),'-');
  if strcmp(xaxis,'log')
    set(gca,'XScale','log');
  end
  hold on;
```

```
  box on;
  grid on;
  xlabel('t');
  ylabel('|Step size|');

  pl = pl+1;
  subplot(npl,1,pl)
  plot(t,q,'-');
  if strcmp(xaxis,'log')
    set(gca,'XScale','log');
  end
  hold on;
  box on;
  grid on;
  xlabel('t');
  ylabel('Order');
end

% Counters
if cntr
  pl = pl+1;
  subplot(npl,1,pl)
  semilogy(t,nst,'k-');
  hold on;
  semilogy(t,nfe,'b-');
  semilogy(t,nni,'r-');
  semilogy(t,netf,'g-');
  semilogy(t,ncfn,'c-');
  if strcmp(xaxis,'log')
    set(gca,'XScale','log');
  end
  box on;
  grid on;
  xlabel('t');
  ylabel('Counters');
end

% Solution components
if sol
  figure(hf2);
  map = colormap;
  ncols = size(map,1);
  hold on;
  for i = 1:N
    hp = plot(t,y(i,:),'-');
    ic = 1+(i-1)*floor(ncols/N);
    set(hp,'Color',map(ic,:));
  end
  if strcmp(xaxis,'log')
    set(gca,'XScale','log');
  end
  box on;
  grid on;
  xlabel('t');
```

```
  ylabel('y');
  title('Solution');
end

drawnow;

%--------------------------------------------------------------------------

function [] = graphical_update(hf1, npl, stats, cntr, sol, dir,...
                                t, h, q, nst, nfe, nni, netf, ncfn,...
                                hf2, y, N)

if npl ~= 0
  figure(hf1);
  pl = 0;
end

% Step size and order
if stats
  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t];
  yd = [get(hc,'YData') abs(h)];
  if length(xd) ~= length(yd)
    disp('h');
  end
  set(hc, 'XData', xd, 'YData', yd);

  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t];
  yd = [get(hc,'YData') q];
  if length(xd) ~= length(yd)
    disp('q');
  end
  set(hc, 'XData', xd, 'YData', yd);
end

% Counters
if cntr
  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  % Attention: Children are loaded in reverse order!
  xd = [get(hc(1),'XData') t];
  yd = [get(hc(1),'YData') ncfn];
  set(hc(1), 'XData', xd, 'YData', yd);
  yd = [get(hc(2),'YData') netf];
  set(hc(2), 'XData', xd, 'YData', yd);
  yd = [get(hc(3),'YData') nni];
  set(hc(3), 'XData', xd, 'YData', yd);
  yd = [get(hc(4),'YData') nfe];
```

```
  set(hc(4), 'XData', xd, 'YData', yd);
  yd = [get(hc(5),'YData') nst];
  set(hc(5), 'XData', xd, 'YData', yd);
end

% Solution components
if sol
  figure(hf2);
  hc = get(gca,'Children');
  xd = [get(hc(1),'XData') t];
  % Attention: Children are loaded in reverse order!
  for i = 1:N
    yd = [get(hc(i),'YData') y(N-i+1,:)];
    set(hc(i), 'XData', xd, 'YData', yd);
  end
end

drawnow;

%----------------------------------------------------------------------

function [] = graphical_final(n, hf1, npl, stats, cntr, sol, dir,...
                              t, h, q, nst, nfe, nni, netf, ncfn,...
                              hf2, y, N, select)

if npl ~= 0
  figure(hf1);
  pl = 0;
end

% Step size and order
if stats
  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n-1)];
  yd = [get(hc,'YData') abs(h(1:n-1))];
  set(hc, 'XData', xd, 'YData', yd);
%  xlim = get(gca,'XLim');
%  set(gca,'XLim',[xlim(1) t(n-1)]);

  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc,'XData') t(1:n-1)];
  yd = [get(hc,'YData') q(1:n-1)];
  set(hc, 'XData', xd, 'YData', yd);
%  xlim = get(gca,'XLim');
%  set(gca,'XLim',[xlim(1) t(n-1)]);
  ylim = get(gca,'YLim');
  set(gca,'YLim',[ylim(1)-1 ylim(2)+1]);
end

% Counters
```

```
if cntr
  pl = pl+1;
  subplot(npl,1,pl)
  hc = get(gca,'Children');
  xd = [get(hc(1),'XData') t(1:n-1)];
  yd = [get(hc(1),'YData') ncfn(1:n-1)];
  set(hc(1), 'XData', xd, 'YData', yd);
  yd = [get(hc(2),'YData') netf(1:n-1)];
  set(hc(2), 'XData', xd, 'YData', yd);
  yd = [get(hc(3),'YData') nni(1:n-1)];
  set(hc(3), 'XData', xd, 'YData', yd);
  yd = [get(hc(4),'YData') nfe(1:n-1)];
  set(hc(4), 'XData', xd, 'YData', yd);
  yd = [get(hc(5),'YData') nst(1:n-1)];
  set(hc(5), 'XData', xd, 'YData', yd);
%  xlim = get(gca,'XLim');
%  set(gca,'XLim',[xlim(1) t(n-1)]);
  legend('nst','nfe','nni','netf','ncfn',2);
end

% Solution components
if sol
  figure(hf2);
  hc = get(gca,'Children');
  xd = [get(hc(1),'XData') t(1:n-1)];
  % Attention: Children are loaded in reverse order!
  for i = 1:N
    yd = [get(hc(i),'YData') y(N-i+1,1:n-1)];
    set(hc(i), 'XData', xd, 'YData', yd);
    cstring{i} = sprintf('y_{%d}',i);
  end
  legend(cstring);
end

drawnow;

%------------------------------------------------------------------------------------------

function [] = text_init(stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

%------------------------------------------------------------------------------------------

function [] = text_update(stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

n = length(t);
for i = 1:n
  if stats
    fprintf('%8.3e %12.6e %1d | ',t(i),h(i),q(i));
  end
  if cntr
    fprintf('%5d %5d %5d %5d %5d\n',nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  else
    fprintf('\n');
  end
```

```
end
fprintf('-----\n');

%------------------------------------------------------------------------

function [] = text_final(n,stats,cntr,t,h,q,nst,nfe,nni,netf,ncfn)

for i = 1:n-1
  if stats
    fprintf('%8.3e %12.6e %1d | ',t(i),h(i),q(i));
  end
  if cntr
    fprintf('%5d %5d %5d %5d %5d\n',nst(i),nfe(i),nni(i),netf(i),ncfn(i));
  else
    fprintf('\n');
  end
end
fprintf('-----\n');
```

2.2 Function types

CVBandJacFn

PURPOSE

CVBandJacFn - type for user provided banded Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

```
CVBandJacFn - type for user provided banded Jacobian function.

IVP Problem

   The function BJACFUN must be defined as
        FUNCTION J = BJACFUN(T,Y,FY)
   and must return a matrix J corresponding to the banded Jacobian of f(t,y).
   The input argument FY contains the current value of f(t,y).
   If a user data structure DATA was specified in CVodeMalloc, then
   BJACFUN must be defined as
        FUNCTION [J, NEW_DATA] = BJACFUN(T,Y,FY,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J,
   the BJACFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

Adjoint Problem

   The function BJACFUNB must be defined either as
        FUNCTION JB = BJACFUNB(T,Y,YB,FYB)
   or as
        FUNCTION [JB, NEW_DATA] = BJACFUNB(T,Y,YB,FYB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc. In either case, it must return the matrix JB, the
   Jacobian of fB(t,y,yB), with respect to yB. The input argument
   FYB contains the current value of f(t,y,yB).

   See also CVodeSetOptions

   See the CVODES user guide for more informaiton on the structure of
   a banded Jacobian.

   NOTE: BJACFUN and BJACFUNB are specified through the property
   JacobianFn to CVodeSetOptions and are used only if the property
   LinearSolver was set to 'Band'.
```

CVDenseJacFn

Purpose

CVDenseJacFn - type for user provided dense Jacobian function.

Synopsis

This is a script file.

Description

CVDenseJacFn - type for user provided dense Jacobian function.

```
IVP Problem

   The function DJACFUN must be defined as
        FUNCTION J = DJACFUN(T,Y,FY)
   and must return a matrix J corresponding to the Jacobian of f(t,y).
   The input argument FY contains the current value of f(t,y).
   If a user data structure DATA was specified in CVodeMalloc, then
   DJACFUN must be defined as
        FUNCTION [J, NEW_DATA] = DJACFUN(T,Y,FY,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J,
   the DJACFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

Adjoint Problem

   The function DJACFUNB must be defined either as
        FUNCTION JB = DJACFUNB(T,Y,YB,FYB)
   or as
        FUNCTION [JB, NEW_DATA] = DJACFUNB(T,Y,YB,FYB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc. In either case, it must return the matrix JB, the
   Jacobian of fB(t,y,yB), with respect to yB. The input argument
   FYB contains the current value of f(t,y,yB).

   See also CVodeSetOptions

   NOTE: DJACFUN and DJACFUNB are specified through the property
   JacobianFn to CVodeSetOptions and are used only if the property
   LinearSolver was set to 'Dense'.
```

CVGcommFn

Purpose

CVGcommFn - type for user provided communication function (BBDPre).

Synopsis

This is a script file.

Description

```
CVGcommFn - type for user provided communication function (BBDPre).

IVP Problem

   The function GCOMFUN must be defined as
        FUNCTION [] = GCOMFUN(T,Y)
   and can be used to perform all interprocess communication necessary
   to evaluate the approximate right-hand side function for the BBDPre
   preconditioner module.
   If a user data structure DATA was specified in CVodeMalloc, then
   GCOMFUN must be defined as
        FUNCTION [NEW_DATA] = GCOMFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then the GCOMFUN function must also
   set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set
   NEW_DATA = DATA as it would lead to unnecessary copying).

Adjoint Problem

   The function GCOMFUNB must be defined either as
        FUNCTION [] = GCOMFUNB(T,Y,YB)
   or as
        FUNCTION [NEW_DATA] = GCOMFUNB(T,Y,YB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc.

   See also CVGlocalFn, CVodeSetOptions

   NOTES:
     GCOMFUN and GCOMFUNB are specified through the GcommFn property in
     CVodeSetOptions and are used only if the property PrecModule is set
     to 'BBDPre'.

     Each call to GCOMFUN is preceded by a call to the RHS function
     ODEFUN with the same arguments T and Y (and YB in the case of GCOMFUNB).
     Thus GCOMFUN can omit any communication done by ODEFUN if relevant
     to the evaluation of G by GLOCFUN. If all necessary communication
     was done by ODEFUN, GCOMFUN need not be provided.
```

CVGlocalFn

Purpose

CVGlocalFn - type for user provided RHS approximation function (BBDPre).

Synopsis

This is a script file.

Description

```
CVGlocalFn - type for user provided RHS approximation function (BBDPre).

IVP Problem
```

```
The function GLOCFUN must be defined as
     FUNCTION G = GLOCFUN(T,Y)
and must return a vector G corresponding to an approximation to f(t,y)
which will be used in the BBDPRE preconditioner module. The case where
G is mathematically identical to F is allowed.
If a user data structure DATA was specified in CVodeMalloc, then
GLOCFUN must be defined as
     FUNCTION [G, NEW_DATA] = GLOCFUN(T,Y,DATA)
If the local modifications to the user data structure are needed
in other user-provided functions then, besides setting the vector G,
the GLOCFUN function must also set NEW_DATA. Otherwise, it should set
NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

Adjoint Problem

The function GLOCFUNB must be defined either as
     FUNCTION GB = GLOCFUNB(T,Y,YB)
or as
     FUNCTION [GB, NEW_DATA] = GLOCFUNB(T,Y,YB,DATA)
depending on whether a user data structure DATA was specified in
CVodeMalloc. In either case, it must return the vector GB
corresponding to an approximation to fB(t,y,yB).

See also CVGcommFn, CVodeSetOptions

NOTE: GLOCFUN and GLOCFUNB are specified through the GlocalFn property
in CVodeSetOptions and are used only if the property PrecModule
is set to 'BBDPre'.
```

CVMonitorFn

Purpose

CVMonitorFn - type for user provided monitoring function.

Synopsis

This is a script file.

Description

```
CVMonitorFn - type for user provided monitoring function.

   The function MONFUN must be defined as
       FUNCTION [] = MONFUN(CALL, T, Y, SSTATS)
   It is called after every internal CVode step and can be used to
   monitor the progress of the solver. MONFUN is called with CALL=0
   from CVodeMalloc at which time it should initialize itself and it
   is called with CALL=2 from CVodeFree. Otherwise, CALL=1.

   It receives as arguments the current time T, solution vector Y,
   and solver statistics structure SSTATS (same as if obtained by
```

```
a call to CVodeGetStats or CVodeGetStatsB).

If additional data is needed inside MONFUN, it must be defined
as
   FUNCTION [] = MONFUN(CALL, T, Y, SSTATS, MONDATA)

A sample monitoring function, CVodeMonitor, is provided with CVODES.

See also CVodeSetOptions, CVodeMonitor

NOTES:

MONFUN is specified through the MonitorFn property in CVodeSetOptions.
If this property is not set, or if it is empty, MONFUN is not used.
MONDATA is specified through the MonitorData property in CVodeSetOptions.
```

CVQuadRhsFn

Purpose

CVQuadRhsFn - type for user provided quadrature RHS function.

Synopsis

This is a script file.

Description

```
CVQuadRhsFn - type for user provided quadrature RHS function.

IVP Problem

   The function ODEQFUN must be defined as
        FUNCTION YQD = ODEQFUN(T,Y)
   and must return a vector YQD corresponding to fQ(t,y), the integrand
   for the integral to be evaluated.
   If a user data structure DATA was specified in CVodeMalloc, then
   ODEQFUN must be defined as
        FUNCTION [YQD, NEW_DATA] = ODEQFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector YQD,
   the ODEQFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

Adjoint Problem

   The function ODEQFUNB must be defined either as
        FUNCTION YQBD = ODEQFUNB(T,Y,YB)
   or as
        FUNCTION [YQBD, NEW_DATA] = ODEQFUNB(T,Y,YB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc. In either case, it must return the vector YQBD
   corresponding to fQB(t,y,yB), the integrand for the integral to be
```

```
   evaluated on the backward phase.

   See also CVodeSetOptions

   NOTE: ODEQFUN and ODEQFUNB are specified through the property
   QuadRhsFn to CVodeSetOptions and are used only if the property
   Quadratures was set to 'on'.
```

CVRhsFn

Purpose

CVRhsFn - type for user provided RHS type

Synopsis

This is a script file.

Description

```
CVRhsFn - type for user provided RHS type

IVP Problem

   The function ODEFUN must be defined as
        FUNCTION YD = ODEFUN(T,Y)
   and must return a vector YD corresponding to f(t,y).
   If a user data structure DATA was specified in CVodeMalloc, then
   ODEFUN must be defined as
        FUNCTION [YD, NEW_DATA] = ODEFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector YD,
   the ODEFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

Adjoint Problem

   The function ODEFUNB must be defined either as
        FUNCTION YBD = ODEFUNB(T,Y,YB)
   or as
        FUNCTION [YBD, NEW_DATA] = ODEFUNB(T,Y,YB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc. In either case, it must return the vector YBD
   corresponding to fB(t,y,yB).

   See also CVodeMalloc, CVodeMallocB

   NOTE: ODEFUN and ODEFUNB are specified through the CVodeMalloc and
   CVodeMallocB functions, respectively.
```

CVRootFn

Purpose

CVRootFn - type for user provided root-finding function.

Synopsis

This is a script file.

Description

CVRootFn - type for user provided root-finding function.

```
   The function ROOTFUN must be defined as
        FUNCTION G = ROOTFUN(T,Y)
   and must return a vector G corresponding to g(t,y).
   If a user data structure DATA was specified in CVodeMalloc, then
   ROOTFUN must be defined as
        FUNCTION [G, NEW_DATA] = ROOTFUN(T,Y,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector G,
   the ROOTFUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   See also CVodeSetOptions

   NOTE: ROOTFUN is specified through the RootsFn property in
   CVodeSetOptions and is used only if the property NumRoots is a
   positive integer.
```

CVSensRhs1Fn

Purpose

CVSensRhs1Fn - type for user provided sensitivity RHS function (single).

Synopsis

This is a script file.

Description

CVSensRhs1Fn - type for user provided sensitivity RHS function (single).

```
   The function ODES1FUN must be defined as
        FUNCTION YSD = ODES1FUN(IS,T,Y,YD,YS)
   and must return a vector YSD corresponding to fS_is(t,y,yS).
   If a user data structure DATA was specified in CVodeMalloc, then
   ODES1FUN must be defined as
        FUNCTION [YSD, NEW_DATA] = ODES1FUN(IS,T,Y,YD,YS,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector YSD,
   the ODES1FUN function must also set NEW_DATA. Otherwise, it should
   set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).
```

```
See also CVodeSetOptions

NOTE: ODES1FUN is specified through the property FSARhsFn to CVodeSetOptions
and is used only if the property SensiAnalysis was set to 'FSA' and if the
property FSARhsType was set to 'One'.
```

CVSensRhsFn

Purpose

CVSensRhsFn - type for user provided sensitivity RHS function.

Synopsis

This is a script file.

Description

CVSensRhsFn - type for user provided sensitivity RHS function.

```
The function ODESFUN must be defined as
     FUNCTION YSD = ODESFUN(T,Y,YD,YS)
and must return a matrix YSD corresponding to fS(t,y,yS).
If a user data structure DATA was specified in CVodeMalloc, then
ODESFUN must be defined as
     FUNCTION [YSD, NEW_DATA] = ODESFUN(T,Y,YD,YS,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the matrix YSD,
the ODESFUN function must also set NEW_DATA. Otherwise, it should
set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

See also CVodeSetOptions

NOTE: ODESFUN is specified through the property FSARhsFn to CVodeSetOptions
and is used only if the property SensiAnalysis was set to 'FSA' and if the
property FSARhsType was set to 'All'.
```

CVJacTimesVecFn

Purpose

CVJacTimesVecFn - type for user provided Jacobian times vector function.

Synopsis

This is a script file.

Description

```
CVJacTimesVecFn - type for user provided Jacobian times vector function.

IVP Problem

   The function JTVFUN must be defined as
        FUNCTION JV = JTVFUN(T,Y,FY,V)
   and must return a vector JV corresponding to the product of the
   Jacobian of f(t,y) with the vector v.
   The input argument FY contains the current value of f(t,y).
   If a user data structure DATA was specified in CVodeMalloc, then
   JTVFUN must be defined as
        FUNCTION [JV, NEW_DATA] = JTVFUN(T,Y,FY,V,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector JV,
   the JTVFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

Adjoint Problem

   The function JTVFUNB must be defined either as
        FUNCTION JVB = JTVFUNB(T,Y,YB,FYB,VB)
   or as
        FUNCTION [JVB, NEW_DATA] = JTVFUNB(T,Y,YB,FYB,VB,DATA)
   depending on whether a user data structure DATA was specified in
   CVodeMalloc. In either case, it must return the vector JVB, the
   product of the Jacobian of fB(t,y,yB) with respect to yB and a vector
   vB. The input argument FYB contains the current value of f(t,y,yB).

   See also CVodeSetOptions

   NOTE: JTVFUN and JTVFUNB are specified through the property
   JacobianFn to CVodeSetOptions and are used only if the property
   LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

CVPrecSetupFn

Purpose

CVPrecSetupFn - type for user provided preconditioner setup function.

Synopsis

This is a script file.

Description

```
CVPrecSetupFn - type for user provided preconditioner setup function.

   The user-supplied preconditioner setup function PSETFUN and
   the user-supplied preconditioner solve function PSOLFUN
   together must define left and right preconditoner matrices
   P1 and P2 (either of which may be trivial), such that the
   product P1*P2 is an approximation to the Newton matrix
```

M = I - gamma*J. Here J is the system Jacobian J = df/dy, and gamma is a scalar proportional to the integration step size h. The solution of systems P z = r, with P = P1 or P2, is to be carried out by the PrecSolve function, and PSETFUN is to do any necessary setup operations.

The user-supplied preconditioner setup function PSETFUN is to evaluate and preprocess any Jacobian-related data needed by the preconditioner solve function PSOLFUN. This might include forming a crude approximate Jacobian, and performing an LU factorization on the resulting approximation to M. This function will not be called in advance of every call to PSOLFUN, but instead will be called only as often as necessary to achieve convergence within the Newton iteration. If the PSOLFUN function needs no preparation, the PSETFUN function need not be provided.

For greater efficiency, the PSETFUN function may save Jacobian-related data and reuse it, rather than generating it from scratch. In this case, it should use the input flag JOK to decide whether to recompute the data, and set the output flag JCUR accordingly.

Each call to the PSETFUN function is preceded by a call to ODEFUN with the same (t,y) arguments. Thus the PSETFUN function can use any auxiliary data that is computed and saved by the ODEFUN function and made accessible to PSETFUN.

IVP Problem

The function PSETFUN must be defined as

```
FUNCTION [JCUR, ERR] = PSETFUN(T,Y,FY,JOK,GAMMA)
```

and must return a logical flag JCUR (true if Jacobian information was recomputed and false if saved data was reused). If PSETFUN was successful, it must return ERR=0. For a recoverable error (in which case the setup will be retried) it must set ERR to a positive integer value. If an unrecoverable error occurs, it must set ERR to a negative value, in which case the integration will be halted. The input argument FY contains the current value of f(t,y). If the input logical flag JOK is false, it means that Jacobian-related data must be recomputed from scratch. If it is true, it means that Jacobian data, if saved from the previous PSETFUN call can be reused (with the current value of GAMMA).

If a user data structure DATA was specified in CVodeMalloc, then PSETFUN must be defined as

```
FUNCTION [JCUR, ERR, NEW_DATA] = PSETFUN(T,Y,FY,JOK,GAMMA,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the flags JCUR and ERR, the PSETFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

Adjoint Problem

```
The function PSETFUNB must be defined either as
     FUNCTION [JCURB, ERR] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB)
or as
     FUNCTION [JCURB, ERR, NEW_DATA] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB,DATA)
depending on whether a user data structure DATA was specified in
CVodeMalloc. In either case, it must return the flags JCURB and ERR.

See also CVPrecSolveFn, CVodeSetOptions

NOTE: PSETFUN and PSETFUNB are specified through the property
PrecSetupFn to CVodeSetOptions and are used only if the property
LinearSolver was set to 'GMRES' or 'BiCGStab' and if the property
PrecType is not 'None'.
```

CVPrecSolveFn

Purpose

CVPrecSolveFn - type for user provided preconditioner solve function.

Synopsis

This is a script file.

Description

CVPrecSolveFn - type for user provided preconditioner solve function.

```
The user-supplied preconditioner solve function PSOLFN
is to solve a linear system P z = r in which the matrix P is
one of the preconditioner matrices P1 or P2, depending on the
type of preconditioning chosen.
```

IVP Problem

```
The function PSOLFUN must be defined as
     FUNCTION [Z, ERR] = PSOLFUN(T,Y,FY,R)
and must return a vector Z containing the solution of Pz=r.
If PSOLFUN was successful, it must return ERR=0. For a recoverable
error (in which case the step will be retried) it must set ERR to a
positive value. If an unrecoverable error occurs, it must set ERR
to a negative value, in which case the integration will be halted.
The input argument FY contains the current value of f(t,y).

If a user data structure DATA was specified in CVodeMalloc, then
PSOLFUN must be defined as
     FUNCTION [Z, ERR, NEW_DATA] = PSOLFUN(T,Y,FY,R,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the vector Z and
the flag ERR, the PSOLFUN function must also set NEW_DATA. Otherwise,
it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would
lead to unnecessary copying).
```

Adjoint Problem

```
The function PSOLFUNB must be defined either as
     FUNCTION [ZB, ERR] = PSOLFUNB(T,Y,YB,FYB,RB)
or as
     FUNCTION [ZB, ERR, NEW_DATA] = PSOLFUNB(T,Y,YB,FYB,RB,DATA)
depending on whether a user data structure DATA was specified in
CVodeMalloc. In either case, it must return the vector ZB and the
flag ERR.

See also CVPrecSetupFn, CVodeSetOptions

NOTE: PSOLFUN and PSOLFUNB are specified through the property
PrecSolveFn to CVodeSetOptions and are used only if the property
LinearSolver was set to 'GMRES' or 'BiCGStab' and if the property
PrecType is not 'None'.
```

3 MATLAB Interface to KINSOL

The MATLAB interface to KINSOL provides access to all functionality of the KINSOL solver.

The interface consists of 5 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions and the types of user-supplied functions are listed in Table 2 and fully documented later in this section. For more in depth details, consult also the KINSOL user guide [1].

To illustrate the use of the KINSOL MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with KINSOL.

Table 2: KINSOL MATLAB interface functions

Functions	KINSetOptions	creates an options structure for KINSOL.
	KINMalloc	allocates and initializes memory for KINSOL.
	KINSol	solves the nonlinear problem.
	KINGetStats	returns statistics for the KINSOL solver.
	KINFree	deallocates memory for the KINSOL solver.
Function types	KINSysFn	system function
	KINDenseJacFn	dense Jacobian function
	KINJactimesVecFn	Jacobian times vector function
	KINPrecSetupFn	preconditioner setup function
	KINPrecSolveFn	preconditioner solve function
	KINGlocalFn	system approximation function (BBDPre)
	KINGcommFn	communication function (BBDPre)

3.1 Interface functions

KINSetOptions

Purpose

KINSetOptions creates an options structure for KINSOL.

Synopsis

```
function options = KINSetOptions(varargin)
```

Description

KINSetOptions creates an options structure for KINSOL.

```
   Usage:

   options = KINSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates a KINSOL
   options structure options in which the named properties have the
   specified values. Any unspecified properties have default values. It is
   sufficient to type only the leading characters that uniquely identify the
   property. Case is ignored for property names.

   options = KINSetOptions(oldoptions,'NAME1',VALUE1,...) alters an existing
   options structure oldoptions.

   options = KINSetOptions(oldoptions,newoptions) combines an existing options
   structure oldoptions with a new options structure newoptions. Any new
   properties overwrite corresponding old properties.

   KINSetOptions with no input arguments displays all property names and their
   possible values.

KINSetOptions properties
(See also the KINSOL User Guide)

MaxNumIter - maximum number of nonlinear iterations [ scalar | 200 ]
   Specifies the maximum number of iterations that the nonlinar solver is allowed
   to take.
MaxNumSetups - [ scalar | 10 ]
   Specifies the maximum number of nonlinear iterations between calls to the
   linear solver setup function (i.e. preconditioner evaluation for an iterative
   linear solver).
MaxNumBetaFails - maximum number of beta-condition failures [ scalar | 10 ]
   Specifies the maximum number of beta-condiiton failures in the line search
   algorithm.
EtaForm - Inexact Newton method [ Constant | Type2 | Type1 ]
   Specifies the method for computing the eta coefficient used in the calculation
   of the linear solver convergence tolerance (used only if strategy='InexactNEwton'
   in the call to KINSol):
      lintol = (eta + eps)*||fscale*f(y)||_L2
   which is the used to check if the following inequality is satisfied:
```

```
     ||fscale*(f(y)+J(y)*p)||_L2 &lt;= lintol
   Valid choices are:
                          | ||f(y_(k+1))||_L2 - ||f(y_k)+J(y_k)*p_k||_L2 |
   EtaForm='Type1'  eta = ------------------------------------------------
                                         ||f(y_k)||_L2

                                  [ ||f(y_(k+1))||_L2 ]^alpha
   EtaForm='Type2'  eta = gamma * [ ----------------- ]
                                  [  ||f(y_k)||_L2    ]
   EtaForm='Constant'
Eta - constant value for eta [ scalar | 0.1 ]
   Specifies the constant value for eta in the case EtaForm='Constant'.
EtaAlpha - alpha parameter for eta [ scalar | 2.0 ]
   Specifies the parameter alpha in the case EtaForm='Type2'
EtaGamma - gamma parameter for eta [ scalar | 0.9 ]
   Specifies the parameter gamma in the case EtaForm='Type2'
MaxNewtonStep - maximum Newton step size [ scalar | 0.0 ]
   Specifies the maximum allowable value of the scaled length of the Newton step.
FuncRelErr - relative residual error [ scalar | eps ]
   Specifies the realative error in computing f(y) when used in difference
   quotient approximation of matrix-vector product J(y)*v.
FuncNormTol - residual stopping criteria [ scalar | eps^(1/3) ]
   Specifies the stopping tolerance on ||fscale*ABS(f(y))||_L-infinity
ScaledStepTol - step size stopping criteria [ scalar | eps^(2/3) ]
   Specifies the stopping tolerance on the maximum scaled step length:
                    ||    y_(k+1) - y_k   ||
                    || ------------------ ||_L-infinity
                    || |y_(k+1)| + yscale ||
InitialSetup - initial call to linear solver setup [ false | true ]
   Specifies whether or not KINSol makes an initial call to the linear solver
   setup function.
MinBoundEps - lower bound on eps [ false | true ]
   Specifies whether or not the value of eps is bounded below by 0.01*FuncNormtol.
Constraints - solution constraints [ vector ]
   Specifies additional constraints on the solution components.
     Constraints(i) =  0 : no constrain on y(i)
     Constraints(i) =  1 : y(i) &gt;= 0
     Constraints(i) = -1 : y(i) &lt;= 0
     Constraints(i) =  2 : y(i) &gt; 0
     Constraints(i) = -2 : y(i) &lt; 0
   If Constraints is not specified, no constraints are applied to y.

LinearSolver - Type of linear solver used [ Dense | BiCGStab | GMRES ]
   Specifies the type of linear solver to be used for the Newton nonlinear solver.
   Valid choices are: Dense (direct, dense Jacobian), GMRES (iterative, scaled
   preconditioned GMRES), BiCGStab (iterative, scaled preconditioned stabilized
   BiCG). The GMRES and BiCGStab are matrix-free linear solvers.
JacobianFn - Jacobian function [ function ]
   This propeerty is overloaded. Set this value to a function that returns
   Jacobian information consistent with the linear solver used (see Linsolver).
   If not specified, KINSOL uses difference quotient approximations.
   For the Dense linear solver, JacobianFn must be of type KINDenseJacFn and must
   return a dense Jacobian matrix. For the iterative linear solvers, GMRES and
   BiCGStab, JacobianFn must be of type KINJactimesVecFn and must return a
```

```
   Jacobian-vector product.
PrecModule - Built-in preconditioner module [ BBDPre | UserDefined ]
   If the PrecModule = 'UserDefined', then the user must provide at least a
   preconditioner solve function (see PrecSolveFn)
   KINSOL provides a built-in preconditioner module, BBDPre which can only be used
   with parallel vectors. It provide a preconditioner matrix that is block-diagonal
   with banded blocks. The blocking corresponds to the distribution of the variable
   vector among the processors. Each preconditioner block is generated from the
   Jacobian of the local part (on the current processor) of a given function g(t,y)
   approximating f(y) (see GlocalFn). The blocks are generated by a difference
   quotient scheme on each processor independently. This scheme utilizes an assumed
   banded structure with given half-bandwidths, mldq and mudq (specified through
   LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian
   block kept by the scheme has half-bandwiths ml and mu (specified through
   LowerBwidth and UpperBwidth), which may be smaller.
PrecSetupFn - Preconditioner setup function [ function ]
   PrecSetupFn specifies an optional function which, together with PrecSolve,
   defines a right preconditioner matrix which is an aproximation
   to the Newton matrix. PrecSetupFn must be of type KINPrecSetupFn.
PrecSolveFn - Preconditioner solve function [ function ]
   PrecSolveFn specifies an optional function which must solve a linear system
   Pz = r, for given r. If PrecSolveFn is not defined, the no preconditioning will
   be used. PrecSolveFn must be of type KINPrecSolveFn.
GlocalFn - Local right-hand side approximation funciton for BBDPre [ function ]
   If PrecModule is BBDPre, GlocalFn specifies a required function that
   evaluates a local approximation to the system function. GlocalFn must
   be of type KINGlocalFn.
GcommFn - Inter-process communication function for BBDPre [ function ]
   If PrecModule is BBDPre, GcommFn specifies an optional function
   to perform any inter-process communication required for the evaluation of
   GlocalFn. GcommFn must be of type KINGcommFn.
KrylovMaxDim - Maximum number of Krylov subspace vectors [ scalar | 10 ]
   Specifies the maximum number of vectors in the Krylov subspace. This property
   is used only if an iterative linear solver, GMRES or BiCGStab, is used
   (see LinSolver).
MaxNumRestarts - Maximum number of GMRES restarts [ scalar | 0 ]
   Specifies the maximum number of times the GMRES (see LinearSolver) solver
   can be restarted.
LowerBwidthDQ - BBDPre preconditioner DQ lower bandwidth [ scalar | 0 ]
   Specifies the lower half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).
UpperBwidthDQ - BBDPre preconditioner DQ upper bandwidth [ scalar | 0 ]
   Specifies the upper half-bandwidth used in the difference-quotient Jacobian
   approximation for the BBDPre preconditioner (see PrecModule).
LowerBwidth - BBDPre preconditioner lower bandwidth [ scalar | 0 ]
   If one of the two iterative linear solvers, GMRES or BiCGStab, is used
   (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used
   (see PrecModule), it specifies the lower half-bandwidth of the retained banded
   approximation of the local Jacobian block.
UpperBwidth - BBDPre preconditioner upper bandwidth [ scalar | 0 ]
   If one of the two iterative linear solvers, GMRES or BiCGStab, is used
   (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used
   (see PrecModule), it specifies the upper half-bandwidth of the retained banded
   approximation of the local Jacobian block.
```

```
See also
     KINDenseJacFn, KINJacTimesVecFn
     KINPrecSetupFn, KINPrecSolveFn
     KINGlocalFn, KINGcommFn
```

KINMalloc

Purpose

KINMalloc allocates and initializes memory for KINSOL.

Synopsis

```
function [] = KINMalloc(fct,n,varargin)
```

Description

KINMalloc allocates and initializes memory for KINSOL.

```
Usage:   KINMalloc ( SYSFUN, N [, OPTIONS [, DATA] ] );

SYSFUN   is a function defining the nonlinearproblem f(y) = 0.
         This function must return a column vector FY containing the
         current value of the residual
N        is the problem dimension.
OPTIONS  is an (optional) set of integration options, created with
         the KINSetOptions function.
DATA     is (optional) problem data passed unmodified to all
         user-provided functions when they are called. For example,
         RES = SYSFUN(Y,DATA).

See also: KINSysFn
```

KINSol

Purpose

KINSol solves the nonlinear problem.

Synopsis

```
function [status,y] = KINSol(y0, strategy, yscale, fscale)
```

Description

KINSol solves the nonlinear problem.

```
Usage:  [STATUS, Y] = KINSol(Y0, STRATEGY, YSCALE, FSCALE)

KINSol manages the computational process of computing an approximate
solution of the nonlinear system. If the initial guess (initial value
assigned to vector Y0) doesn't violate any user-defined constraints,
```

```
then KINSol attempts to solve the system f(y)=0. If an iterative linear
solver was specified (see KINSetOptions), KINSol uses a nonlinear Krylov
subspace projection method. The Newton-Krylov iterations are stopped
if either of the following conditions is satisfied:

    ||f(y)||_L-infinity &lt;= 0.01*fnormtol

    ||y[i+1] - y[i]||_L-infinity &lt;= scsteptol

However, if the current iterate satisfies the second stopping
criterion, it doesn't necessarily mean an approximate solution
has been found since the algorithm may have stalled, or the
user-specified step tolerance may be too large.

STRATEGY specifies the global strategy applied to the Newton step if it is
unsatisfactory. Valid choices are 'None' or 'LineSearch'.
YSCALE is a vector containing diagonal elements of scaling matrix for vector
Y chosen so that the components of YSCALE*Y (as a matrix multiplication) all
have about the same magnitude when Y is close to a root of f(y)
FSCALE is a vector containing diagonal elements of scaling matrix for f(y)
chosen so that the components of FSCALE*f(Y) (as a matrix multiplication)
all have roughly the same magnitude when u is not too near a root of f(y)

On return, status is one of the following:
  0: KINSol succeeded
  1: The initial y0 already satisfies the stopping criterion given above
  2: Stopping tolerance on scaled step length satisfied
 -1: Illegal attempt to call before KINMalloc
 -2: One of the inputs to KINSol is illegal.
 -5: The line search algorithm was unable to find an iterate sufficiently
     distinct from the current iterate
 -6: The maximum number of nonlinear iterations has been reached
 -7: Five consecutive steps have been taken that satisfy the following
     inequality:
          ||yscale*p||_L2 &gt; 0.99*mxnewtstep
 -8: The line search algorithm failed to satisfy the beta-condition
     for too many times.
 -9: The linear solver's solve routine failed in a recoverable manner,
     but the linear solver is up to date.
-10: The linear solver's intialization routine failed.
-11: The linear solver's setup routine failed in an unrecoverable manner.
-12: The linear solver's solve routine failed in an unrecoverable manner.

See also KINSetOptions, KINGetstats
```

KINGetStats

Purpose

KINGetStats returns statistics for the main KINSOL solver and the linear

Synopsis

function si = KINGetStats()

Description

KINGetStats returns statistics for the main KINSOL solver and the linear
solver used.

```
   Usage: solver_stats = KINGetStats;

Fields in the structure solver_stats

o nfe    - total number evaluations of the nonlinear system function SYSFUN
o nni    - total number of nonlinear iterations
o nbcf   - total number of beta-condition failures
o nbops  - total number of backtrack operations (step length adjustments)
           performed by the line search algorithm
o fnorm  - scaled norm of the nonlinear system function f(y) evaluated at the
           current iterate: ||fscale*f(y)||_L2
o step   - scaled norm (or length) of the step used during the previous
           iteration: ||uscale*p||_L2
o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

  Fields in LSinfo for the 'Dense' linear solver

o name - 'Dense'
o njeD - number of Jacobian evaluations
o nfeD - number of right-hand side function evaluations for difference-quotient
         Jacobian approximation

  Fields in LSinfo for the 'GMRES' or 'BiCGStab' linear solver

o name - 'GMRES' or 'BiCGStab'
o nli  - number of linear solver iterations
o npe  - number of preconditioner setups
o nps  - number of preconditioner solve function calls
o ncfl - number of linear system convergence test failures
```

KINFree

Purpose

KINFree deallocates memory for the KINSOL solver.

Synopsis

function [] = KINFree()

Description

KINFree deallocates memory for the KINSOL solver.

```
   Usage:  KINFree
```

3.2 Function types

KINDenseJacFn

Purpose

KINDenseJacFn - type for user provided dense Jacobian function.

Synopsis

This is a script file.

Description

KINDenseJacFn - type for user provided dense Jacobian function.

```
   The function DJACFUN must be defined as
        FUNCTION [J, IER] = DJACFUN(Y,FY)
   and must return a matrix J corresponding to the Jacobian of f(y).
   The input argument FY contains the current value of f(y).
   If successful, IER should be set to 0. If an error occurs, IER should
   be set to a nonzero value.
   If a user data structure DATA was specified in KINMalloc, then
   DJACFUN must be defined as
        FUNCTION [J, IER, NEW_DATA] = DJACFUN(Y,FY,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the matrix J and
   the flag IER, the DJACFUN function must also set NEW_DATA. Otherwise,
   it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
   to unnecessary copying).

   See also KINSetOptions

   NOTE: DJACFUN is specified through the property JacobianFn to KINSetOptions
   and is used only if the property LinearSolver was set to 'Dense'.
```

KINGcommFn

Purpose

KINGcommFn - type for user provided communication function (BBDPre).

Synopsis

This is a script file.

Description

KINGcommFn - type for user provided communication function (BBDPre).

```
   The function GCOMFUN must be defined as
        FUNCTION [] = GCOMFUN(Y)
   and can be used to perform all interprocess communication necessary
   to evaluate the approximate right-hand side function for the BBDPre
```

```
   preconditioner module.
   If a user data structure DATA was specified in KINMalloc, then
   GCOMFUN must be defined as
        FUNCTION [NEW_DATA] = GCOMFUN(Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then the GCOMFUN function must also
   set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set
   NEW_DATA = DATA as it would lead to unnecessary copying).

   See also KINGlocalFn, KINSetOptions

   NOTES:
     GCOMFUN is specified through the GcommFn property in KINSetOptions
     and is used only if the property PrecModule is set to 'BBDPre'.

     Each call to GCOMFUN is preceded by a call to the system function
     SYSFUN with the same argument Y. Thus GCOMFUN can omit any communication
     done by SYSFUN if relevant to the evaluation of G by GLOCFUN. If all
     necessary communication was done by SYSFUN, GCOMFUN need not be provided.
```

KINGlocalFn

Purpose

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

Synopsis

This is a script file.

Description

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

```
   The function GLOCFUN must be defined as
        FUNCTION G = GLOCFUN(Y)
   and must return a vector G corresponding to an approximation to f(y)
   which will be used in the BBDPRE preconditioner module. The case where
   G is mathematically identical to F is allowed.
   If a user data structure DATA was specified in KINMalloc, then
   GLOCFUN must be defined as
        FUNCTION [G, NEW_DATA] = GLOCFUN(Y,DATA)
   If the local modifications to the user data structure are needed
   in other user-provided functions then, besides setting the vector G,
   the GLOCFUN function must also set NEW_DATA. Otherwise, it should set
   NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   See also KINGcommFn, KINSetOptions

   NOTE: GLOCFUN is specified through the GlocalFn property in KINSetOptions
   and is used only if the property PrecModule is set to 'BBDPre'.
```

KINJacTimesVecFn

Purpose

KINJacTimesVecFn - type for user provided Jacobian times vector function.

Synopsis

This is a script file.

Description

KINJacTimesVecFn - type for user provided Jacobian times vector function.

```
   The function JTVFUN must be defined as
        FUNCTION [JV, FLAG, IER] = JTVFUN(Y,V,FLAG)
   and must return a vector JV corresponding to the product of the
   Jacobian of f(y) with the vector v. On input, FLAG indicates if
   the iterate has been updated in the interim. JV must be update
   or reevaluated, if appropriate, unless FLAG=false. This flag must
   be reset by the user.
   If successful, IER should be set to 0. If an error occurs, IER should
   be set to a nonzero value.
   If a user data structure DATA was specified in KINMalloc, then
   JTVFUN must be defined as
        FUNCTION [JV, FLAG, IER, NEW_DATA] = JTVFUN(Y,V,FLAG,DATA)
   If the local modifications to the user data structure are needed in
   other user-provided functions then, besides setting the vector JV, and
   flags FLAG and IER, the JTVFUN function must also set NEW_DATA. Otherwise,
   it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
   unnecessary copying).

   See also KINSetOptions

   NOTE: JTVFUN is specified through the property JacobianFn to KINSetOptions
   and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINPrecSetupFn

Purpose

KINPrecSetupFn - type for user provided preconditioner setup function.

Synopsis

This is a script file.

Description

KINPrecSetupFn - type for user provided preconditioner setup function.

```
  The user-supplied preconditioner setup subroutine should compute
  the right-preconditioner matrix P used to form the scaled preconditioned
  linear system:
```

```
(Df*J(y)*(P^-1)*(Dy^-1)) * (Dy*P*x) = Df*(-F(y))

where Dy and Df denote the diagonal scaling matrices whose diagonal elements
are stored in the vectors YSCALE and FSCALE, respectively.

The preconditioner setup routine (referenced by iterative linear
solver modules via pset (type KINSpilsPrecSetupFn)) will not be
called prior to every call made to the psolve function, but will
instead be called only as often as necessary to achieve convergence
of the Newton iteration.

Note: If the PRECSOLVE function requires no preparation, then a
preconditioner setup function need not be given.

 The function PSETFUN must be defined as
      FUNCTION [IER] = PSETFUN(Y,YSCALE,FY,FSCALE)
 If successful, PSETFUN must return IER=0. If an error occurs, then IER
 must be set to a non-zero value.
 The input argument FY contains the current value of f(y), while YSCALE
 and FSCALE are the scaling vectors for solution and system function,
 respectively (as passed to KINSol)

 If a user data structure DATA was specified in KINMalloc, then
 PSETFUN must be defined as
      FUNCTION [IER, NEW_DATA] = PSETFUN(Y,YSCALE,FY,FSCALE,DATA)
 If the local modifications to the user data structure are needed in
 other user-provided functions then, besides setting the flag IER,
 the PSETFUN function must also set NEW_DATA. Otherwise, it should
 set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead
 to unnecessary copying).

 See also KINPrecSolveFn, KINSetOptions, KINSol

 NOTE: PSETFUN is specified through the property PrecSetupFn to KINSetOptions
 and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINPrecSolveFn

Purpose

KINPrecSolveFn - type for user provided preconditioner solve function.

Synopsis

This is a script file.

Description

KINPrecSolveFn - type for user provided preconditioner solve function.

```
 The user-supplied preconditioner solve function PSOLFN
 is to solve a linear system P z = r in which the matrix P is
 the preconditioner matrix (possibly set implicitely by PSETFUN)
```

```
The function PSOLFUN must be defined as
     FUNCTION [Z,IER] = PSOLFUN(Y,YSCALE,FY,FSCALE,R)
and must return a vector Z containing the solution of Pz=r.
If successful, PSOLFUN must return IER=0. If an error occurs, then IER
must be set to a non-zero value.
The input argument FY contains the current value of f(y), while YSCALE
and FSCALE are the scaling vectors for solution and system function,
respectively (as passed to KINSol)

If a user data structure DATA was specified in KINMalloc, then
PSOLFUN must be defined as
     FUNCTION [Z, IER, NEW_DATA] = PSOLFUN(Y,YSCALE,FY,FSCALE,R,DATA)
If the local modifications to the user data structure are needed in
other user-provided functions then, besides setting the vector Z and
the flag IER, the PSOLFUN function must also set NEW_DATA. Otherwise,
it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would
lead to unnecessary copying).

See also KINPrecSetupFn, KINSetOptions

NOTE: PSOLFUN is specified through the property PrecSolveFn to KINSetOptions
and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.
```

KINSysFn

PURPOSE

KINSysFn - type for user provided system function

SYNOPSIS

This is a script file.

DESCRIPTION

KINSysFn - type for user provided system function

```
The function SYSFUN must be defined as
     FUNCTION FY = SYSFUN(Y)
and must return a vector FY corresponding to f(y).
If a user data structure DATA was specified in KINMalloc, then
SYSFUN must be defined as
     FUNCTION [FY, NEW_DATA] = SYSFUN(Y,DATA)
If the local modifications to the user data structure are needed
in other user-provided functions then, besides setting the vector FY,
the SYSFUN function must also set NEW_DATA. Otherwise, it should set
NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to
unnecessary copying).

See also KINMalloc

NOTE: SYSFUN is specified through the KINMalloc function.
```

4 Supporting modules

This section describes two additional modules in sundialsTB, nvector and putils. The functions in nvector perform various operations on vectors. For serial vectors, all of these operations default to the corresponding matlab functions. For parallel vectors, they can be used either on the local portion of the distributed vector or on the global vector (in which case they will trigger an MPI Allreduce operation). The functions in putils are used to run parallel sundialsTB applications. The user should only call the function `mpirun` to launch a parallel matlab application. See one of the paralel sundialsTB examples for usage.

The functions in these two additional modules are listed in Table 3 and described in detail in the remainder of this section.

Table 3: The nvector and putils functions

NVECTOR	N_VMax	returns the largest element of x
	N_VMaxNorm	returns the maximum norm of x
	N_VMin	returns the smallest element of x
	N_VDotProd	returns the dot product of two vectors
	N_VWrmsNorm	returns the weighted root mean square norm of x
	N_VWL2Norm	returns the weighted Euclidean L2 norm of x
	N_VL1Norm	returns the L1 norm of x
PUTILS	mpirun	runs parallel examples
	mpiruns	runs the parallel example on a child matlab process
	LAM_Start	lamboot and `MPI_Init` master (if required)
	LAM_Finish	clean mpiTB mex files from memory

4.1 NVECTOR functions

N_VDotProd

PURPOSE

```
N_VDotProd returns the dot product of two vectors
```

SYNOPSIS

```
function ret = N_VDotProd(x,y,comm)
```

DESCRIPTION

```
N_VDotProd returns the dot product of two vectors

   Usage:  RET = N_VDotProd ( X, Y [, COMM] )

If COMM is not present, N_VDotProd returns the dot product of the
local portions of X and Y. Otherwise, it returns the global dot
product.
```

SOURCE CODE

```
function ret = N_VDotProd(x,y,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$


if nargin == 2

  ret = dot(x,y);

else

  ldot = dot(x,y);
  gdot = 0.0;
  MPI_Allreduce(ldot,gdot,'SUM',comm);
  ret = gdot;

end
```

N_VL1Norm

PURPOSE

```
N_VL1Norm returns the L1 norm of x
```

SYNOPSIS

```
function ret = N_VL1Norm(x,comm)
```

DESCRIPTION

```
N_VL1Norm returns the L1 norm of x

   Usage:  RET = N_VL1Norm ( X [, COMM] )

If COMM is not present, N_VL1Norm returns the L1 norm of
the local portion of X. Otherwise, it returns the global
L1 norm..
```

Source Code

```
function ret = N_VL1Norm(x,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

if nargin == 1

  ret = norm(x,1);

else

  lnrm = norm(x,1);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'MAX',comm);
  ret = gnrm;

end
```

N_VMax

Purpose

```
N_VMax returns the largest element of x
```

Synopsis

```
function ret = N_VMax(x,comm)
```

Description

```
N_VMax returns the largest element of x

   Usage:  RET = N_VMax ( X [, COMM] )

If COMM is not present, N_VMax returns the maximum value of
the local portion of X. Otherwise, it returns the global
maximum value.
```

Source Code

```
function ret = N_VMax(x,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$
```

```
if nargin == 1

  ret = max(x);

else

  lmax = max(x);
  gmax = 0.0;
  MPI_Allreduce(lmax,gmax,'MAX',comm);
  ret = gmax;

end
```

N_VMaxNorm

Purpose

N_VMaxNorm returns the L-infinity norm of x

Synopsis

function ret = N_VMaxNorm(x, comm)

Description

N_VMaxNorm returns the L-infinity norm of x

```
   Usage:  RET = N_VMaxNorm ( X [, COMM] )
```

If COMM is not present, N_VMaxNorm returns the L-infinity norm of the local portion of X. Otherwise, it returns the global L-infinity norm..

Source Code

```
function ret = N_VMaxNorm(x, comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

if nargin == 1

  ret = norm(x,'inf');

else

  lnrm = norm(x,'inf');
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'MAX',comm);
  ret = gnrm;

end
```

N_VMin

PURPOSE

N_VMin returns the smallest element of x

SYNOPSIS

function ret = N_VMin(x,comm)

DESCRIPTION

```
N_VMin returns the smallest element of x
   Usage:  RET = N_VMin ( X [, COMM] )

If COMM is not present, N_VMin returns the minimum value of
the local portion of X. Otherwise, it returns the global
minimum value.
```

SOURCE CODE

```
function ret = N_VMin(x,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

if nargin == 1

  ret = min(x);

else

  lmin = min(x);
  gmin = 0.0;
  MPI_Allreduce(lmin,gmin,'MIN',comm);
  ret = gmin;

end
```

N_VWL2Norm

PURPOSE

N_VWL2Norm returns the weighted Euclidean L2 norm of x

SYNOPSIS

function ret = N_VWL2Norm(x,w,comm)

DESCRIPTION

N_VWL2Norm returns the weighted Euclidean L2 norm of x
with weight vector w:

```
sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)]

Usage:  RET = N_VWL2Norm ( X, W [, COMM] )
```

If COMM is not present, N_VWL2Norm returns the weighted L2 norm of the local portion of X. Otherwise, it returns the global weighted L2 norm..

Source Code

```
function ret = N_VWL2Norm(x,w,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

if nargin == 2

  ret = dot(x.^2,w.^2);
  ret = sqrt(ret);

else

  lnrm = dot(x.^2,w.^2);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'SUM',comm);

  ret = sqrt(gnrm);

end
```

N_VWrmsNorm

Purpose

N_VWrmsNorm returns the weighted root mean square norm of x

Synopsis

```
function ret = N_VWrmsNorm(x,w,comm)
```

Description

N_VWrmsNorm returns the weighted root mean square norm of x
with weight vector w:

```
sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)/N]

Usage:  RET = N_VWrmsNorm ( X, W [, COMM] )
```

If COMM is not present, N_VWrmsNorm returns the WRMS norm of the local portion of X. Otherwise, it returns the global WRMS norm..

Source Code

```
function ret = N_VWrmsNorm(x,w,comm)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

if nargin == 2

  ret = dot(x.^2,w.^2);
  ret = sqrt(ret/length(x));

else

  lnrm = dot(x.^2,w.^2);
  gnrm = 0.0;
  MPI_Allreduce(lnrm,gnrm,'SUM',comm);

  ln = length(x);
  gn = 0;
  MPI_Allreduce(ln,gn,'SUM',comm);

  ret = sqrt(gnrm/gn);

end
```

4.2 Parallel utilities

---

mpirun

PURPOSE

mpirun runs parallel examples.

SYNOPSIS

function [] = mpirun(fct,npe,dbg)

DESCRIPTION

```
 mpirun runs parallel examples.

   Usage:  mpirun ( FCT , NPE [, DBG] )

   FCT - name (or handle) of the function to be executed on all MATLAB
         processes.
   NPE - number of processes to be used (including the master).
   DBG - flag for debugging. If true, spawn MATLAB child processes
         with a visible xterm. (default DBG=false)
```

SOURCE CODE

```
function [] = mpirun(fct,npe,dbg)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

ih = isa(fct,'function_handle');
is = isa(fct,'char');
if ih
  sh = functions(fct);
  fct_str = sh.function;
elseif is
  fct_str = fct;
else
  error('mpirun:: Unrecognized function');
end

if exist(fct_str) ~= 2
  err_msg = sprintf('mpirun:: Function %s not in search path.',fct_str);
  error(err_msg);
end

nslaves = npe-1;
LAM_Start(nslaves);

debug = false;
if (nargin > 2) & dbg
  debug = true;
```

```
end

cmd_slaves = sprintf('mpiruns(''%s'')',fct_str);

if debug
  cmd = 'xterm';
  args = {'-e','matlab','-nosplash','-nojvm','-r',cmd_slaves};
else
  cmd = 'matlab';
  args = {'-nosplash','-nojvm','-r',cmd_slaves};
end

[info children errs] = MPI_Comm_spawn(cmd,args,nslaves,'NULL',0,'SELF');

[info NEWORLD] = MPI_Intercomm_merge(children,0);

nvm(1,NEWORLD);
feval(fct,NEWORLD);
nvm(2);

LAM_Finish;
```

mpiruns

Purpose

`mpiruns` runs the parallel example on a child MATLAB process.

Synopsis

`function [] = mpiruns(fct)`

Description

```
mpiruns runs the parallel example on a child MATLAB process.
  This function should not be called directly. It is called
  by mpirun on the spawned child processes.
```

Source Code

```
function [] = mpiruns(fct)

% Radu Serban <radu@llnl.gov>
% Copyright (c) 2005, The Regents of the University of California.
% $Revision$Date$

[dum hostname]=system('hostname');
fprintf('child MATLAB process on %s\n',hostname);

MPI_Init;

MPI_Errhandler_set('WORLD','RETURN');

[info parent] = MPI_Comm_get_parent;

```

```
fprintf('waiting for the parent to merge MPI intercommunicators ... ');
[info NEWORLD] = MPI_Intercomm_merge(parent,1);
fprintf('OK!\n');

MPI_Errhandler_set(NEWORLD,'RETURN');

nvm(1,NEWORLD);
feval(fct,NEWORLD);
nvm(2);

MPI_Finalize;
LAM_Finish;
```

LAM_Finish

Purpose

LAM_Finish cleans MPITB MEX files from memory.

Synopsis

function LAM_Finish

Description

```
 LAM_Finish cleans MPITB MEX files from memory.

  Most probably used in the following sequence:

  MPI_Init
  ...
  &lt;MPITB code&gt;
  ...
  MPI_Finalize;
  LAM_Clean;        % required to avoid
  MPI_Init;        % matlab crash due to MPI re-init

  See MPI_Init help page for more details
```

Source Code

```
function LAM_Finish

[M, MEX] = inmem;                    % clear all MPI_* MEX files
M = MEX(strmatch('MPI_',MEX));
clear(M{:})                          % allow for MPI_Init again
```

LAM_Start

Purpose

LAM_Start invokes lamboot (if required) and MPI_Init (if required).

Synopsis

function LAM_Start(nslaves, rpi, hosts)

Description

```
LAM_Start invokes lamboot (if required) and MPI_Init (if required).

  Usage: LAM_Init [ ( NSLAVES [, RPI [, HOSTS] ] ) ]

  LAM_Start boots LAM and initializes MPI to match a given number of slave
  hosts (and rpi) from a given list of hosts. All three args optional.

  If they are not defined, HOSTS are taken from a builtin HOSTS list
  (edit HOSTS at the beginning of LAM_Start.m to match your cluster)
  or from the bhost file if defined through LAMBHOST (in this order).

  If not defined, RPI is taken from the builtin variable RPI (edit it
  to suit your needs) or from the LAM_MPI_SSI_rpi environment variable
  (in this order).
```

Source Code

```
function LAM_Start(nslaves, rpi, hosts)

% Heavily based on the LAM_Init function in MPITB.

%----------------------------------------------------------
% DEFAULT VALUES %
%----------------------------------------------------------

HOSTS = {'tux30','tux76','tux105','tux111'};
RPI   = 'tcp';

%----------------------------------------------------------
% ARGCHECK
%----------------------------------------------------------

%% List of hosts

if nargin>2
  % hosts passed as an argument...
  if ~iscell(hosts)
    error('LAM_Init: 3rd arg is not a cell');
  else
    for i=1:length(hosts)
      if ~ischar(hosts{i})
        error('LAM_Init: 3rd arg is not cell-of-strings');
      end
    end
  end
else
  % We must get the hosts from somewhere else...
  if ~isempty(HOSTS)
    hosts = HOSTS;                        % Variable HOSTS defined above
  else
    bfile = getenv('LAMBHOST');
    if ~isempty(bfile)
      hosts = readHosts(bfile);           % bhost defined in environment
    else
      % Cannot define hosts!
```

```
      error('LAM_Init:: cannot find list of hosts');
    end
  end
end

%% RPI

if nargin>1
  % RPI passed as an argument
  if ~ischar(rpi)
    error('LAM_Init: 2nd arg is not a string')
  else
    % full rpi name, if single letter used
    rpi=rpi_str(rpi);
    if isempty(rpi)
      error('LAM_Init: 2nd arg is not a known RPI')
    end
  end
else
  % We must get RPI from somewhere else...
  if ~isempty(RPI)
    rpi = rpi_str(RPI);                         % Variable RPI defined above
  else
    RPI = getenv('LAM_MPI_SSI_rpi');
    if ~isempty(RPI)
      rpi = rpi_str(RPI);                       % RPI defined in environment
    else
      error('LAM_Init:: cannot find RPI');
    end
  end
end

% Number of slaves

if nargin>0
  if ~isreal(nslaves) || fix(nslaves)~=nslaves  || nslaves>=length(hosts)
    error('LAM_Init: 1st arg is not a valid #slaves')
  end
else
  nslaves = length(hosts)-1;
end

%-------------------------------------
% LAMHALT %
%--------------------------------------------------------------------------
% reasons to lamhalt:
% - not enough nodes  (nslv+1) % NHL < NSLAVES+1
% - localhost not      in list % weird - just lamboot (NHL=0)
% - localhost not last in list % weird - just lamboot (NHL=0)
%--------------------------------------------------------------------------

% Lam Nodes Output
[stat, LNO] = system('lamnodes');
if ~stat                                        % already lambooted
```

```
  emptyflag = false;
  if isempty(LNO)
    % this shouldn't happen
    emptyflag=true;
    % it's MATLAB's fault I think
    fprintf('pushing stubborn MATLAB "system" call (lamnodes): ');
  end

  while isempty(LNO) || stat
    fprintf('.');
    [stat, LNO] = system('lamnodes');
  end
  if emptyflag
    fprintf('\n');
  end

  LF = char(10);
  LNO = split(LNO,LF);                          % split lines in rows at \n

  [stat, NHL] = system('lamnodes|wc -l');       % Number of Hosts in Lamnodes

  emptyflag = false;                            % again,
  if isempty(NHL)                               % this shouldn't happen
    emptyflag=true;                             % it's MATLAB's fault I think
    fprintf('pushing stubborn MATLAB "system" call (lamnodes|wc): ');
  end
  while isempty(NHL) || stat
    fprintf('.');
    [stat, NHL] = system('lamnodes|wc -l');
  end
  if emptyflag
    fprintf('\n');
  end

  NHL = str2num(NHL);
  if NHL ~= size(LNO,1) || ~ NHL>0              % Oh my, logic error
    NHL= 0;                                     % pretend there are no nodes
    disp('LAM_Init: internal logic error: lamboot')
  end                                           % to force lamboot w/o lamhalt
  if isempty(findstr(LNO(end,:),'this_node'))   % master computer last in list
    disp('LAM_Init: local host is not last in nodelist, hope that''s right')
    beforeflag=0;
    for i=1:size(LNO,1)
      if ~isempty(findstr(LNO(i,:),'this_node'))
        beforeflag=1;
        break;                                  % well, not 1st but it's there
      end
    end                                         % we already warned the user
    if ~beforeflag                              % Oh my, incredible, not there
      NHL= 0;                                   % pretend there are no nodes
      disp('LAM_Init: local host not in LAM? lamboot')
    end
  end                                           % to force lamboot w/o lamhalt
```

```

  if NHL > 0                                  % accurately account multiprocessors
    NCL = 0;                                  % number of CPUs in lamnodes
    for i=1:size(LNO,1)                       % add the 2nd ":"-separated
      fields=split(LNO(i,:),':');             % field, ie, #CPUs
      NCL = NCL + str2num(fields(2,:));
    end
    if NCL<NHL                                % Oh my, logic error
      NHL= 0;                                 % pretend there are no nodes
      disp('LAM_Init: internal logic error: lamboot')
    else
      % update count
      NHL=NCL;
    end                                       % can't get count from MPI,
  end                                         % since might be not _Init'ed

  if NHL < nslaves+1                          % we have to lamboot

    % but avoid getting caught
    [infI flgI]=MPI_Initialized;              % Init?
    [infF flgF]=MPI_Finalized;                % Finalize?
    if infI ||  infF
      error('LAM_Init: error calling _Initialized/_Finalized?')
    end
    if flgI && ~flgF                          % avoid hangup due to
      MPI_Finalize;                           % inminent lamhalt
      clear MPI_*                             % force MPI_Init in Mast/Ping
      disp('LAM_Init: MPI already used- clearing before lamboot')
    end                                       % by pretending "not _Init"
    if NHL > 0                                % avoid lamhalt in weird cases
      disp('LAM_Init: halting LAM')
      system('lamhalt');                      % won't get caught on this
    end
  end
end

%-----------------------------
% LAMBOOT
%------------------------------------------------------------
% reasons to lamboot:          %
% - not lambooted yet          % stat~=0
% - lamhalted above (or weird) % NHL < NSLAVES+1 (0 _is_ <)
%------------------------------------------------------------

if stat || NHL<nslaves+1

  HNAMS=hosts{end};
  for i=nslaves:-1:1
    HNAMS=strvcat(hosts{i},HNAMS);
  end
  HNAMS = HNAMS';                             % transpose for "for"

  fid=fopen('bhost','wt');
  for h = HNAMS
```

```
      fprintf(fid,'%s\n',h');                        % write slaves' hostnames
    end
    fclose(fid);
    disp   ('LAM_Init: booting LAM')

    stat = system('lamboot -s -v bhost');

    if stat                                          % again, this shouldn't happen
      fprintf('pushing stubborn MATLAB "system" call (lamboot): ');
      while stat
        fprintf('.'); stat = system('lamboot -s -v bhost');
      end
      fprintf('\n');
    end

    system('rm -f bhost');                           % don't need bhost anymore
end                                                  % won't wipe on exit/could lamhalt

%------------------------------------------
% RPI CHECK
%------------------------------------------

[infI flgI] = MPI_Initialized;                       % Init?
[infF flgF] = MPI_Finalized;                         % Finalize?

if infI || infF
  error('LAM_Start: error calling  Initialized/ Finalized?')
end

if   flgI && ~flgF                                   % Perfect, ready to start
else                                                 % something we could fix?
  if flgI ||  flgF                                   % MPI used, will break
    clear MPI_*                                      % unless we clear MPITB
    disp('LAM_Start: MPI already used- clearing')  % must start over
  end

  MPI_Init;
end

%------------------------------------------
% NSLAVES CHECK
%------------------------------------------

[info attr flag] = MPI_Attr_get(MPI_COMM_WORLD,MPI_UNIVERSE_SIZE);
if info | ~flag
  error('LAM_Init: attribute MPI_UNIVERSE_SIZE does not exist?')
end
if attr<2
  error('LAM_Init: required 2 computers in LAM')
end

%==========================================================================

function rpi = rpi_str(c)
```

```
%RPI_STR Full LAM SSI RPI string given initial letter(s)
%
%  rpi = rpi_str (c)
%
%  c     initial char(s) of rpi name: t,l,u,s
%  rpi   full rpi name, one of: tcp, lamd, usysv, sysv
%        Use '' if c doesn't match to any supported rpi
%

flag = nargin~=1 || isempty(c) || ~ischar(c);
if flag
  return
end

c=lower(c(1));
rpis={'tcp','lamd','usysv','sysv','none'};     % 'none' is sentinel

for i=1:length(rpis)
  if rpis{i}(1)==c
    break
  end
end

if i<length(rpis)
  rpi=rpis{i};                                  % normal cases
else
  rpi='';                                       % no way, unknown rpi
end
```

References

[1] A. M. Collier, A. C. Hindmarsh, R. Serban, and C.S. Woodward. User Documentation for KINSOL v2.2.0. Technical Report UCRL-SM-208116, LLNL, 2004.

[2] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS, suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, (submitted), 2004.

[3] A. C. Hindmarsh and R. Serban. User Documentation for CVODES v2.1.0. Technical report, LLNL, 2004. UCRL-SM-208111.